AFRL-RI-RS-TR-2007-273
In-House Final Technical Report
January 2008

HIGH PERFORMANCE COMPUTING (HPC) FOR REAL-TIME COURSE OF ACTION (COA) ANALYSIS

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2007-273 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/ /s/

MICHAEL J. HAYDUK, Chief
Emerging Computing Tech Branch
Advanced Computing Division

JAMES A. COLLINS, Deputy Chief
Advanced Computing Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JAN 2008	Final	Nov 03 – Aug 07

4. TITLE AND SUBTITLE

HIGH PERFORMANCE COMPUTING (HPC) FOR REAL-TIME COURSE OF ACTION (COA) ANALYSIS

5a. CONTRACT NUMBER
In-House

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Duane A. Gilmour and William E. McKeever

5d. PROJECT NUMBER
RTCO

5e. TASK NUMBER
A0

5f. WORK UNIT NUMBER
04

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFRL/RITC
525 Brooks Rd
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RITC
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2007-273

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 07-0693

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The focus of this research project was to develop and demonstrate proof of concept technologies to assist decision makers in assessing friendly effects-based Course of Actions (COAs) against an intelligent adversary in an operational-level simulation environment, faster than real-time. The R&D activities included multiple components: a simulation test bed; a scalable, flexible simulation framework; automated scenario generation techniques with dynamic update; intelligent adversarial behavior modeling; effects-based/attrition-based behavior modeling; and real-time analysis technology for comparing and grading effectiveness of alternative simulations. The following highlights the in-house accomplishments: developed an effects-based center of gravity (COG) modeling capability and an in-house COA simulation test bed capable of simulating direct, indirect, cumulative, cascading and recovery events. The following complimentary capabilities were also developed under extramural research projects: semi-automated scenario generation that produced COAs in minutes/hours vs days, simulation cloning on HPCs for parallel COA evaluation, dynamic COA analysis incorporating an unscripted adversary, and COA simulation comparisons produced in seconds vs hours.

15. SUBJECT TERMS
Effects-based operations, wargaming, scenario generation, center of gravity, course of action, adversary modeling, measure of effectiveness, measure of performance, modeling and simulation, Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Duane A. Gilmour
U	U	U	UL	72	19b. TELEPHONE NUMBER (*Include area code*) N/A

Table of Contents

List of Figures

List of Tables

1. Executive Summary

The military planning process, which involves a series of complex decisions in an uncertain environment, is highly manpower intensive and is carried out continuously throughout a campaign. Plans and strategies, which result in courses of action (COAs), are evaluated to determine the necessary steps to meet the overall strategic objectives. Currently, COAs are evaluated by two techniques. One technique involves teams of individuals playing both sides of a campaign, while trying to predict the outcome based on each others actions. This technique is manpower intensive, and cannot be maintained at the speed of current operations, and also doesn't allow for sufficient "what-if" COA analysis. The second technique involves automated wargaming technologies. Automated techniques are faster; however, they are performed against a scripted adversary and focus on attrition based modeling. They are incapable of assessing effects and their contribution to the overall mission objectives, which is inherent in effects based operations (EBO). The focus of this research project was to develop and demonstrate proof of concept technologies to assist decision makers in assessing friendly effects based COAs against an intelligent adversary in an operational-level simulation environment, faster than real-time. The benefits to the warfighter include, but are not limited to the following:

- Multiple ranked effects based plans.
- Greater understanding of the mission space (past, present, future).
- Anticipation of the adversary resulting in an ability to continuously shape dynamic situations.
- Interactive capability to conduct theater/campaign-level "what-if" analysis.

The research and development activities included multiple components: a simulation test bed; a scalable, flexible simulation framework; automated scenario generation techniques with dynamic update; intelligent adversarial behavior modeling; effects based/attrition based behavior modeling; and real-time analysis technology for comparing and grading effectiveness of alternative simulations. The following highlights the accomplishments of the research project:

- The in-house research team accomplished the following:
    - Researched, developed and demonstrated effects based center of gravity (COG) modeling capability.
    - Developed an in-house force structure simulation (FSS) test bed capable of simulating direct, indirect, cumulative, cascading, and recovery events (military vs. military and infrastructure).
    - Transitioned the in-house developed COG modeling capability to a Small Business Innovative Research (SBIR) contractor.
    - Delivered the FSS test bed to several contractors to support further research and development (R&D).
    - Interfaced capabilities/technologies with other Air Force Research Laboratory (AFRL) developed technologies, including the Strategy Development Tool (SDT), Athena and COG-a.
- Complementary research as a result of the in-house effort:

    - Semi-automated scenario generation capability produced COAs in minutes/hours vs. days.
    - Simulation cloning on high performance computers (HPC) for parallel COA evaluation.
    - Dynamic COA/enemy COA (eCOA) analysis – demonstration of dynamic COA analysis incorporating unscripted adversary actions.
    - COA simulation comparisons produced in seconds vs. hours.
- Authored/co-authored 11 technical papers (see Section 6)

All of these research components were pursued in parallel; however, many components were integrated together to help solidify research approaches and for demonstration purposes, with FSS being the central component. The intent of the project was to demonstrate (successfully) the concepts supporting a dynamic effects based COA analysis capability, not to produce an end system.

2. Introduction

The military planning process depends upon analysis systems to help planners anticipate and respond in real-time to a dynamically changing battlespace with counteractions. Complex technical challenges exist in developing automated processes to derive hypotheses about future alternatives for mission scenarios. The military conducts combat operations in the presence of uncertainty and the alternatives that might emerge. It is virtually impossible to identify or predict the specific details of what might transpire. Current generation wargaming technologies typically execute a pre-scripted sequence of events for an adversary, independent of the opposing force actions. A significant research challenge for wargaming is predicting and assessing how friendly actions result in adversary behavioral outcomes, and how those behavioral outcomes impact the adversary commander's decisions and future actions.

The focus of this research was to develop technologies to assist decision makers in assessing friendly COAs against an operational-level adversarial environment. Utilizing HPC technology, it is possible to dynamically execute multiple simulations concurrently to evaluate COAs for critical elements related to execution and timing as well as overall effectiveness against a range of adversarial, or eCOA. Conventional automated wargames are also insufficient when it comes to evaluating modern campaign approaches. They focus on traditional attrition based force-on-force modeling, whereas modern campaign strategies employ and evaluate a mixture of kinetic and non-kinetic operations, such as effects based operations. EBO is an approach to planning, executing and assessing military operations that focuses on obtaining a desired strategic outcome or "effect" on the adversary instead of merely attacking targets or simply dealing with objectives. For wargames to be effective, they must allow users to evaluate multiple ways to accomplish the same goal with a combination of direct, indirect, complex, cumulative, and cascading effects. The overarching objective of this research activity was to address the challenges of simulating EBO COAs in the presence of a dynamic adversarial environment, faster than real-time. Such a system would allow planners to evaluate the effectiveness of today's alternative decisions and plans in tomorrow's battlefield. The approach pursued under this research project was as follows:

- Develop an HPC simulation research test bed to support in-house and contractor R&D.

- Develop generic effects based modeling approach.
- Use high performance computing for rapid assessment of concurrent scenarios.
- Develop adversary modeling approaches for predictive enemy COAs.
- Develop techniques to automate scenario generation from stored and live data.
- Investigate approaches for spawning COAs for automated "what-if" analysis.

3. Research Program

While much of the research was performed extramurally by contractors, the close relationship between AFRL researchers and these organizations resulted in a synergistic research project. The following sections of this report will focus mainly on the in-house research aspects of the project; and will only touch briefly on the extramural research to provide the complete picture. References are included for all of the extramurally funded research, to include final technical reports and conference papers.

Figure 1 represents the architecture for the research project. Research supporting the FSS test bed and the Effects Based/COG Modeling was performed in-house at AFRL. Securboration, Inc. performed research associated with Automated Scenario Generation and Emergent Adversary Behavior Modeling; Stottler Henke Associates, Inc. also performed research on Emergent Adversary Behavior Modeling; Metron, Inc. researched techniques in Rapid Decision Branch Analysis and Science Applications International Corporation (SAIC) performed research on the Course of Action Simulation Analysis (CASA) capabilities.

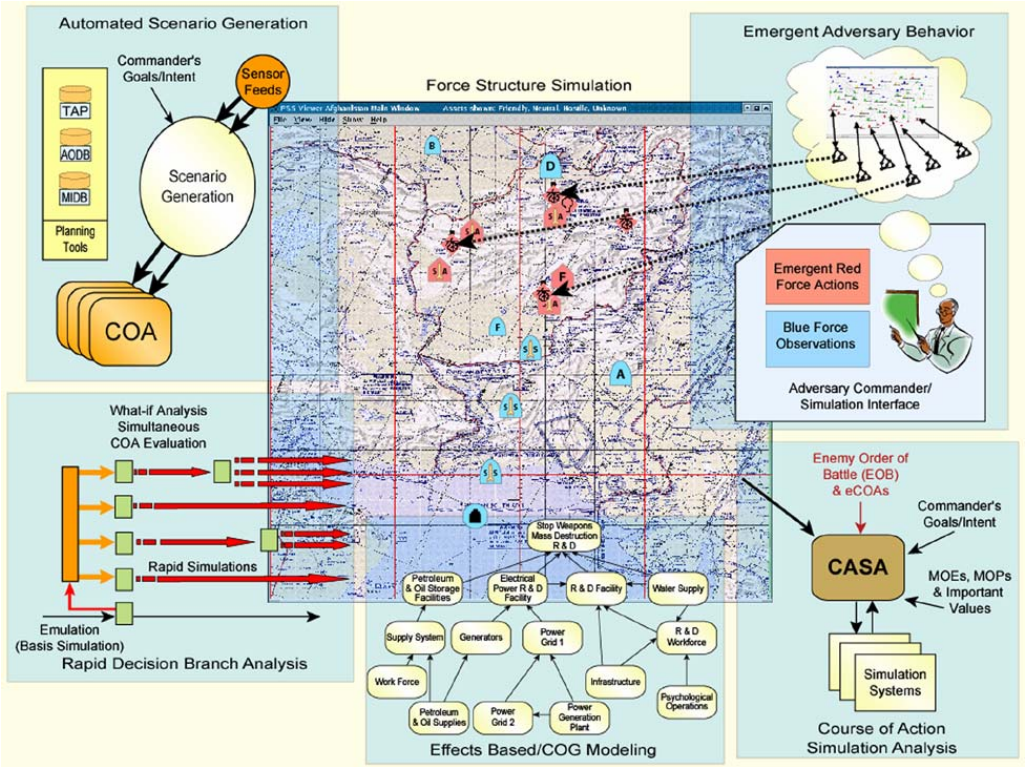

Figure 1. Real-Time Decision Support Architecture

3.1 Force Structure Simulation

The foundation for the research and development activities was an in-house FSS research test bed. FSS was developed solely to support the development and integration of various supporting technologies relating to COA/eCOA comparison. This in-house test bed provides a means by which researchers can modify the simulation environment to support the implementation and prototyping of a variety of related research activities. FSS runs on the Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) framework. SPEEDES was chosen as the foundation for FSS because it helps exploit available high performance computational resources and provides much needed functionality. SPEEDES distributes and coordinates simulations across multiple central processing units of various HPC architectures; including workstations, clusters, or combinations of architectures. FSS is a discrete event simulation meaning there is no 'time step' to the simulation. Simulation proceeds to the next event, not the next time step, and the events drive subsequent simulation action. The simulation objects are high level architecture (HLA) objects and the simulation can be federated with other simulations through the use of a SPEEDES external module.

A range of widely used wargaming frameworks were examined before deciding to employ the SPEEDES framework. SPEEDES is a discrete event driven simulation with conservative and optimistic schemes of operation. All of the communications employed by SPEEDES are standard, architecture independent operations which enable the simulation to compile and run on a variety of HPC platforms. The framework supports C++ class structures. SPEEDES has been successfully used as the discrete event simulation framework by a number of wargaming environments.

FSS currently extends force-on-force simulation to include cascading and cumulative effects associated with EBO. The research accomplished to extend FSS from a traditional force-on-force simulator to one that includes effects, is further discussed in Section 3.2. Simulation objects include Assets, Commanders, and Abstract Center of Gravity objects. Assets may be fixed or mobile, while Commanders are only mobile. Abstract Centers of Gravity enable the simulation of a work force, or an abstract attribute like morale. Mobile Assets include surface-to-air missile (SAM) assets, missile assets, air assets, naval assets, vehicle assets, environmental assets and teams.

FSS is an operational level simulator. Sub-units of assets, such as the flight deck of an aircraft carrier, are not independently supported. It uses a Monte Carlo simulation approach, but executes only a single run, where the user can specify the seed for the random number generator.

The random numbers are used in three areas: probability of an indirect effect cascading, probability of hitting a target, and the fraction of damage induced on a target that is hit. A uniform distribution is used for the random numbers.

Currently, FSS models the following:

- Several types of objects in the simulation:
    - Fixed assets (airbase, bunkers, bridges, intersections, CommandPosts).
    - Mobile assets.

        - Aircraft: F-15, A-10, unmanned aerial vehicle (UAV), Mig-21, B-2, F117, FA18.
        - NavalAsset: BattleGroup.
        - SAM (Gadfly, Gammon, SA-10, SA-5, SA-2, Anti Aircraft Artillery (AAA)).
        - Missiles: Scud, Seersuckers, Exocets.
        - Vehicle: truck.
        - Team: Suicide bombing teams, multiple rocket launchers.
    - Commanders.
    - Abstract EBO characteristics (e.g. leadership/morale/work force).
- Scripted command sequences for assets:
    - Retasking.
    - Scripts must exist at the beginning of simulation.
    - A SPEEDES external module can be used to interact with an external intelligent adversary allowing assets to diverge from their original script.
- Probability of hitting a target (weapon dependence)
- Randomized damage from a weapon's effect
- Model interdependency (indirect and cascading effects of EBO):
    - Fixed assets.
    - Mobile assets.
    - Commanders.
    - Abstract EBO characteristics (e.g. leadership/morale/work force).
- Sensor range
- Low-observable assets
- Various weapon classes:
    - Active sensor.
    - Global Positioning System (GPS).
    - Weapons of mass destruction (WMD).
    - Soft weapons such as leaflets for propaganda.
    - Jamming of Command & Control (C2).
- Automatic engagement modes for mobile assets:
    - Combat air patrol (CAP) (Air to Air).
    - Suppression of Enemy Air Defenses (SEAD) (air against SAM).
    - AntiScud (air against missile assets).
    - Interdiction (all mobile assets against any enemy mobile assets).
    - SAM assets engage all enemy air assets when they 'operate'.

Currently, FSS does not model the following:

- Fuel consumption or maximum ranges of assets.
- Transport of materiel (moving petroleum, oil and lubricants (POL); sub assets/armaments).
- Maximum speed or altitude of assets.
- Minimum speed of assets.
- Terrain or line-of sight.
- Weather effects on performance (this can be accomplished by using an external module).
- Weapon Target interactions (Assume that weapon target pairing has been done successfully).

3.2 Effects Based/COG Modeling

The desire to simulate EBO characteristics as a behavioral object within a wargaming environment required the conventional attrition-based FSS framework to be extended in several ways. The simulation had to be capable of simulating not only the standard direct kinetic events but also non-kinetic actions and indirect cascading events and complex interactions that control the state of specific COGs. Furthermore, a method to observe the simulation objects and obtain indicators related to the state of the EBO object was required. A Java tool, referred to as "JavaCOG," was developed to support the creation of EBO COG models and their related properties, developing an abstract COA and producing the parameter file that the simulation required. The following sections will discuss the construction of the EBO object, the event structure, how the simulation responds to the events, and the JavaCOG tool.

3.2.1 Simulation Framework

The simulation framework was developed using C++ and the SPEEDES application programming interface (API). Every EBO simulation object is created using the same base object class. In using this generic class approach, any object can exhibit EBO characteristics. The simulation object properties are initialized and constructed by parsing an input parameter file. This parameter file contains the properties that define the simulation object including name, types of actions, the action's properties, and the other EBO objects it depends on, including their cascading event properties. A cascading event is defined here as any property of a dependent node that will have an effect on the object. The EBO object creates its own specific state structure by creating an array of possible actions and properties associated with those actions. The simulation object also obtains the names of objects it relies upon; it receives events from those objects using the objects name as a trigger string. The object then creates an array of all the dependant objects along with possible cascading event properties. Object properties will be discussed in more detail below.

3.2.1.1 Event Framework

There are several types of events in the SPEEDES framework that were used in the simulation. This section will explain what types of events were used, why they were selected and how the events function. The first type of event is a point to point event where a simulation object schedules an event on another specific object. This event scheme is used to schedule the COA actions on an object or scan an object for an indicator. The next event type is used to allow an object to schedule an event on itself and control its internal state, for example, powering off a generator as fuel is exhausted. These events are referred to as local events. When a simulation

object is affected by a cascading event or a COA event, the object calculates future actions/reactions and schedules an event on itself to notify when the new state begins.

Since the design approach taken was to have a dynamically coupled network, it's essential for an object to schedule events on multiple unknown objects. This final event type used for implementing cascading events is much more complex. A cascading event is scheduled on objects that rely on an affected object and the relation matrix for scheduling those events are unknown and arbitrarily complex. The SPEEDES undirected event was well suited to accomplish this task. Undirected events are scheduled using a trigger string. The event is broadcast to all simulation objects subscribed to the specific trigger string. Therefore, when an objects state change creates a cascading event, the object will schedule an event with its name as the trigger string and the properties affected as the data. The restriction implied here is that each EBO objects must have a unique name.

3.2.1.2 Event Processing

The simulation objects must react to different types of events in different ways. As stated above there are many types of events that are necessary to simulate EBO object behavior. This section illustrates how the simulation object will react when it receives any of the events.

When a simulation object receives a COA event, the object identifies the type of action and then finds the properties associated with that COA action. The simulation object then calculates the impact of the action using a random number and the states probability properties to determine if the COA will have an effect on the object. If the COA fails to produce an effect, the object continues operating normally. But, if the COA action produces an effect, the simulation object updates its potential action list and schedules two local events. The first event scheduled controls the amount of time delay between the COA action and when the objects state may change, the effect. The scheduled time for the affected event is calculated by acquiring the current time then adding the delay time of the influencing action. The second local event is scheduled for when the simulation object would recover from the direct action. This event is scheduled by adding the recovery time to the current time plus the delay time.

The EBO object assesses its potential state change by testing to determine if any events were received that would cause this new state change to be ignored. There are two reasons that could cause the state to be ignored. The first is that the affect has already occurred and the object is already in that state or a new influencing factor eliminates the need for a state change. For example, if the backup system was repaired before the delay time occurred then the object would remain operational. When all checks have been verified, the object schedules a cascading event to all of the objects that have subscribed to its trigger string to inform those objects that it has been affected. When the simulation object receives a cascading event, the EBO object repeats the same analysis and behavior processes that were performed when the COA event was detected, as described above. This type of event ripples through the simulation from object to object.

The ability of a simulation object to recover from an affected state is an important feature that was also implemented. An object can attempt to recover from an affected state after it receives one out of two events. The first type of event is a local event that was scheduled on the object itself after it has been affected by a direct action. The second event is a cascading event that signals the elimination of the indirect effect event that may change the objects current state. These two

different recovery methods are both handled in virtually the same manner, and the object updates its array to reflect the repair. To find the status of the object, the object evaluates its 'objects and actions' arrays looking for an affected object in the object array or an affected action in the action array. If all of the objects and actions are normal, then the object can change its state from affected to operational. However, if there is an object still affecting it, the main object will reevaluate its own state to determine if it is still being affected. If the object can operate without the affected sub-object, then the state changes. If it is still affected by an object or an action, then the current state will remain. If the simulation objects operational status has changed, then it must produce an event to inform the objects that depend on it of the new status. The cascading recovery event is accomplished in a similar manner as the cascading event.

The last event type received by an object is a direct event to scan/query the EBO object in an attempt to observe the objects indicators. The simulation processes a scan action by printing indicators depending on the state. However, before it prints out an indicator, it will check if the indicator can be observed. Certain indicators can only be observed between certain hours of the day. The EBO object will try to print its indicator, and then evaluate all of the property values for the objects it depends on, by checking if they are operational, and checking to see if the indicator (either operating or influenced) can be printed out. By printing the indicators in this manner, it is possible to get mixed indicators and therefore, more realistic indicators.

3.2.2 JavaCOG – A COG authoring tool

JavaCOG is a Java-based authoring tool that has three main functions. It serves as a graphical way to develop COG models and extends the models to include EBO characteristics. The authoring tool can be used to define an abstract COA to use in analyzing the COG system developed. The last function is to generate the parameter file that is required to build and simulate the COG system and apply the abstract COA for testing and analysis.

JavaCOG is comprised of three windows, as seen in Figure 2. The window on the right, the main window, is the graphical way to represent the model. The window on the left is a tree representation of the nodes and the possible actions that can be scheduled or actions that have been scheduled. The bottom window is a summary of the current abstract COA in a table format with additional debugging columns.

Properties are edited in the main window. Some properties are predefined as cascading properties. The cascading property is defined in the relationship between the nodes and the directions of the arrows. The node at the tail of the arrow relies directly on the node at the head of the arrow. In Figure 2, the Power Grid relies directly on the Power Plant. If the Power Plant is affected, then the Power Grid could be affected by a cascading event.

The user can further define a node's properties by utilizing the editing mode. Figure 3 is an example of the screen a user would see when editing the Work Force node. The top half of this screen allows the user to change the name of the node, edit the actions, and edit the indicators for this node. On the lower half of the screen are the properties of all the nodes that can affect the Work Force. The Influencing Attribute drop down menu allows the user to select the influencing node to edit. The user can then select the probability that this influencing node will cause the main node to be affected. For example, in Figure 3, if the Transportation Infrastructure node was

affected, there would be a high probability that the Work Force node would be affected in a cascading event. Currently, the probabilities values are set from 0.2 (very low) to 1.0 (very high), in increments of 0.2. Next, the user has the option to assign a complex effect for the influencing node. A complex effect is defined where all of the influencing nodes must be affected to influence the main node. This translates to in this example, for the Work Force to be influenced; the Transportation Infrastructure and another node must be influenced. In this example, there is no complex effect for the Work Force node. The next property that can be edited is the delay time. The delay time is the amount of time it takes for the action to directly influence the main node. The last properties to be edited are the indicators. The indicators in this section are in reference to the observations that can be made at the main node (Work Force) that would indicate the state of the influencing node (Transportation Infrastructure).

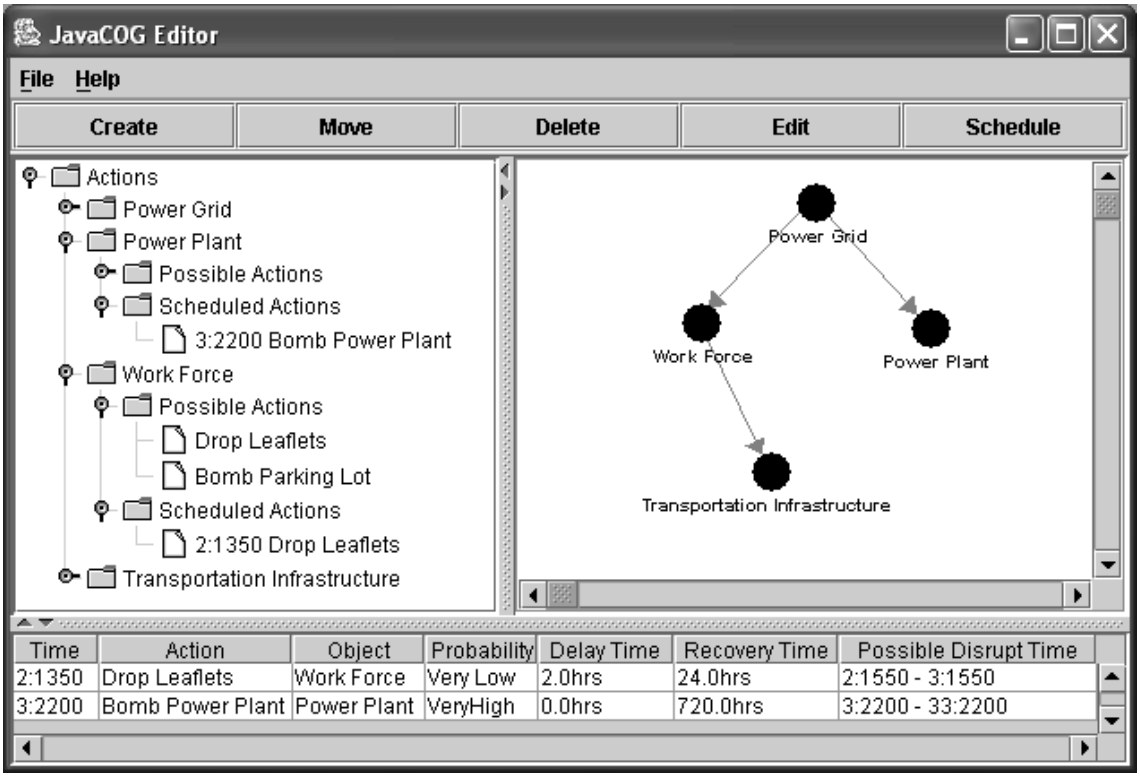

Figure 2. Example of JavaCOG

A fundamental concept of EBO is the ability to influence the enemy by using kinetic and non-kinetic tactics. A simulation object must be able to react to different actions in different ways. Therefore, an object must know what type of actions can be taken against it, and how to react. When modifying or creating an action, the user will have a window similar to Figure 4. From this window the user can enter the properties that the simulation will need, such as the name of the action, the delay time, the probability of success, and the amount of time it will it take for the object to recover from this action.

Once the COG model is complete and a COA test set has been developed, the next step is to simulate the COA. The JavaCOG tool has been developed to generate a SPEEDES parameter file which can be utilized within FSS.

The simulation of EBO in a wargaming environment has been accomplished through the integration of COG models into the FSS test bed. The COG models support indirect, cascading and complex effects, which are an essential characteristic of EBO.

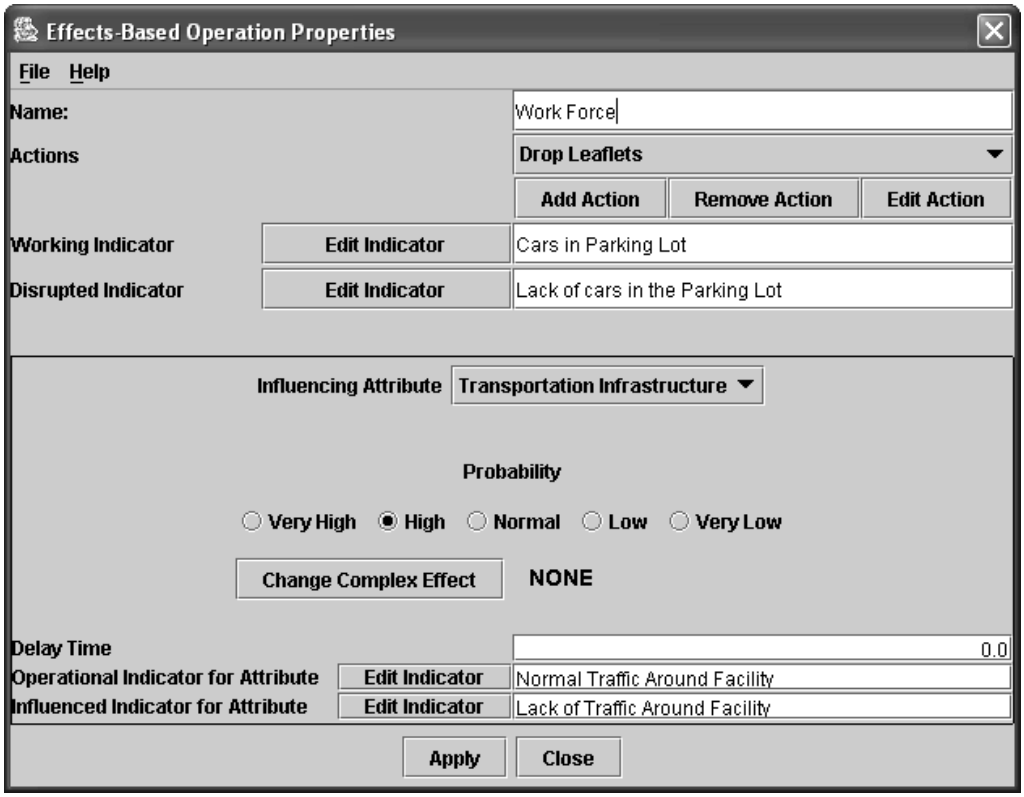

Figure 3. Example of Editing Work Force

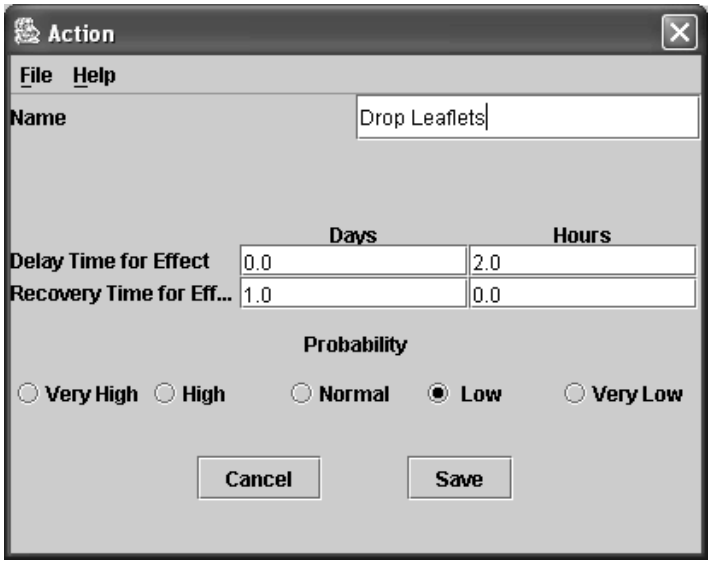

Figure 4. Example of Action Properties

3.2.3 Integrating COG Models into FSS

The COG modeling methodology provides the framework necessary for simulating EBO concepts. One of the key EBO concepts is the cascading event. This simulation event represents the cascading nature of effects, which occur when a direct effect "ripples through an enemy target system, often influencing other target systems as well", resulting in an indirect effect or outcome. In the wargame, this occurs when one simulation object is influenced by another simulation object that it relies upon. For example, if a factory is dependent on a power plant to function, then an event that causes the power plant to be disabled will cascade to the factory causing the factory to shutdown. A second essential EBO concept is the complex effect. This type of effect reflects the cumulative nature of effects. For example, the production capability of a factory could be halted by destroying numerous transfer stations and generators, which are necessary for the power plant to function. A third key EBO concept is the center of gravity. COGs that are interdependencies of assets, such as the factory and the numerous transfer stations and power plants could be simulated in a force-on-force simulation that includes indirect and cascading effects. But not all COGs are an interdependency of assets. Some COGs can include more abstract concepts such as a "Work Force", or Leadership or morale.

To transform FSS from an attrition based force-on-force simulator to an effects based simulator, abstract COG elements such as morale and indirect and cascading events needed to be integrated within the simulation framework. To enable the simulation of indirect and cascading events, two new classes were required in FSS. The first class, "BaseCOG", controls the event scheduling, event handling, and logic for the simulation of effects based actions. The other class, "effects", is a data class that BaseCOG uses to determine how to process EBO events. Using the inheritance and polymorphism properties of object-oriented programming, capturing the ability to simulate indirect and cascading events was straightforward. The class diagram from the attrition based version of FSS is shown in Figure 5. FSS had two main types of simulation objects, assets and commanders, which allowed FSS to model and simulate attrition type force-on-force COAs. The addition of the BaseCOG and the effects classes to the FSS class structure resulted in a simulation capability for indirect and cascading effects. This is due to inheritance. The placement of the BaseCOG class above the assets and commanders classes in the class structure of the COG modeling methodology allows any simulation object to inherit from this class. And, since the BaseCOG class represents effects based properties, the inheriting classes will, as well. This class structure also allows the simulation objects to interact with any other simulation object. This interaction is necessary for representing the interrelationships inherent in COG models. The new class diagram is shown in Figure 6.

When FSS was an attrition based simulation, the class structure described in Figure 5 was suitable. Commanders would give scripts (orders) to assets, assets had a location, and assets would only attack other assets directly. Now that FSS is capable of simulating indirect and cascading events, the wargame needed to be expanded to model abstract concepts. For instance, if a commander wanted to wargame a COA to affect Troop A's morale, then the simulation would need to know Troop A's morale object type. A new simulation object was introduced into FSS, called AbstractCOG to allow for abstract concepts, such as "World Opinion" to be modeled. The AbstractCOG can also represent an entity that is comprised of many locations, but could be modeled as one (e.g. Work Force). The AbstractCOG inherits directly from the BaseCOG, as

shown in Figure 6, and it will have all the properties needed to implement EBO concepts, and will not be bounded by the constraints that an asset must follow.

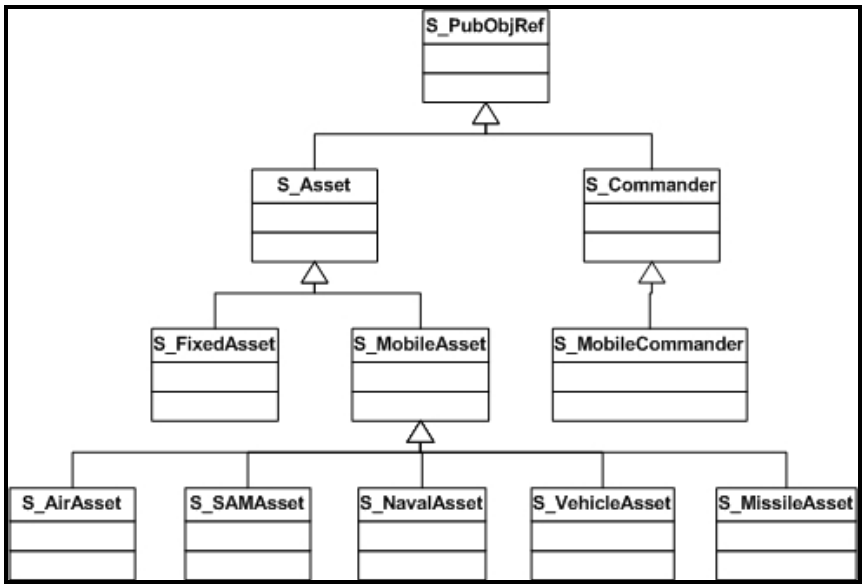

Figure 5. FSS Class Diagram before the Introduction of EBO

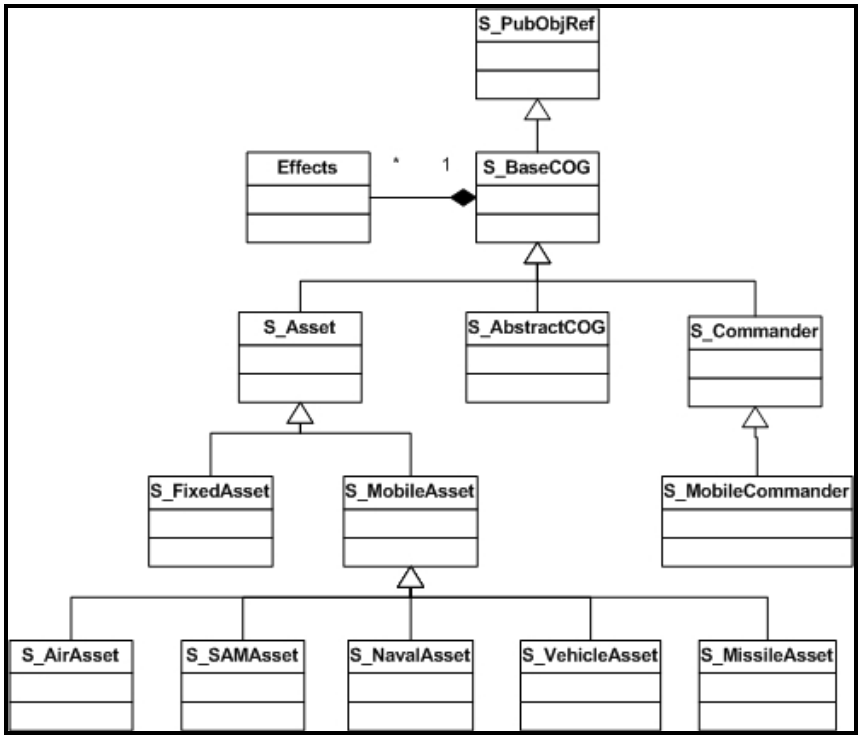

Figure 6. Class Diagram of FSS with EBO Capabilities

With the addition of the BaseCOG class to FSS to implement indirect and cascading events, came the requirement to be able to illustrate the status of the simulation objects. This is necessary to

understand if the indirect, complex and cascading effects are being achieved. The current version of FSS has three possible states: operational, disrupted, and destroyed. The COG modeling methodology allows for layers of indicators to reflect the possible states. There is a top level indicator, which reflects the overall status of the object. A top level indicator of a factory being affected could be: an infrared scan of the factory has detected no heat signatures from machines or people. There are also sub-level indicators, which illustrate if a cascading event had an affect on the object. An example of a sub-level indicator for a factory that has been affected might be that the work force is not present, evident by the employees' parking lot being empty during operating hours. For both levels of indicators, there is observable evidence to determine whether the object is operational or disabled.

The integration of the COG modeling methodology and the AbstractCOG class within FSS extended the simulation capability of FSS to encompass EBO concepts. This extension provides FSS with the capability of simulating direct, indirect, complex and cascading effects, which is more accurate and representative of current and future operations. Furthermore, FSS can provide several types of indicators, which provides a mechanism for the user to observe the current state of any simulation object, and the impact of the aforementioned effects. While FSS is capable of modeling abstract concepts such as "World Opinion", understanding how those concepts should be modeled and the impact of those concepts requires further research.

3.2.4 Simulating EBO

A simple scenario was created to exercise and demonstrate the concept of indirect and cascading effects that are intrinsic to EBO. This scenario was also used to demonstrate that COG models can be used to circumvent erroneous behavior in some simulations. The scenario can be defined by three parameter files: an assets file, a commander's file and a COG file. In this section the scenario is described, and the results of the simulations are presented and compared.

Some erroneous behavior can occur in FSS when assets that house commanders are destroyed. In FSS there are three types of simulation objects: assets, commanders and abstract COGs. Assets in the simulation execute missions which they receive from their commanders. However, within FSS, assets may only execute missions on other assets and not on their commanders. Suppose there is a commander at a blue Airborne Command Post, directing blue assets in an area of interest, and the red forces destroy the blue Airborne Command Post. The aircraft would be destroyed, but the commander would continue to provide scripted missions to its other assets. This behavior can provide an erroneous evaluation of the scenario being simulated. A mechanism that links a commander to an asset so that when the asset is destroyed, the commander stops functioning, would alleviate this situation and result in a much better simulation. Implementation of the COG dependency modeling mechanism overcomes this erroneous behavior.

Exploiting indirect effects and the COG modeling capability, it is possible to link a commander with an asset. This is accomplished by creating a dependency description that has the specified commander depending on the simulation asset, (a Command Post or an Airbase). The effect of disabling the Airbase or Command Post by engaging it with some assets, cascades into a disabling of the commander.

The geographical context of the simple, fictional, scenario is shown in Figure 7, where there are three red power plants, two red airbases, a red bunker, and a blue battle group. The objective of the simple scenario is to demonstrate that the simulator supports direct, indirect, cascading and complex effects. This is achieved by disabling Airbase 2 and two Air Defense Commanders, as shown in the COG model in Figure 8. Four aircraft are launched from the blue battle group to engage the red bunker and the three power plants. The red commander (Air Defense Commander 2) has a SAM asset that can be deployed against the blue aircraft, unless the red commander is disabled through complex and cascading effects.

Figure 8 shows the COG relationship for this simple example, which specifies assets and commanders. Air Defense Commander 2 depends on Airbase 2, which in turn depends on the three power plants. Airbase 1 depends on the two red commanders, Air Defense Commander 1 and Air Defense Commander 2. Air Defense Commander 1 is resident in Bunker ID_19 and is dependent on that bunker. In this scenario, the aircraft attack the bunker and then the power plants.

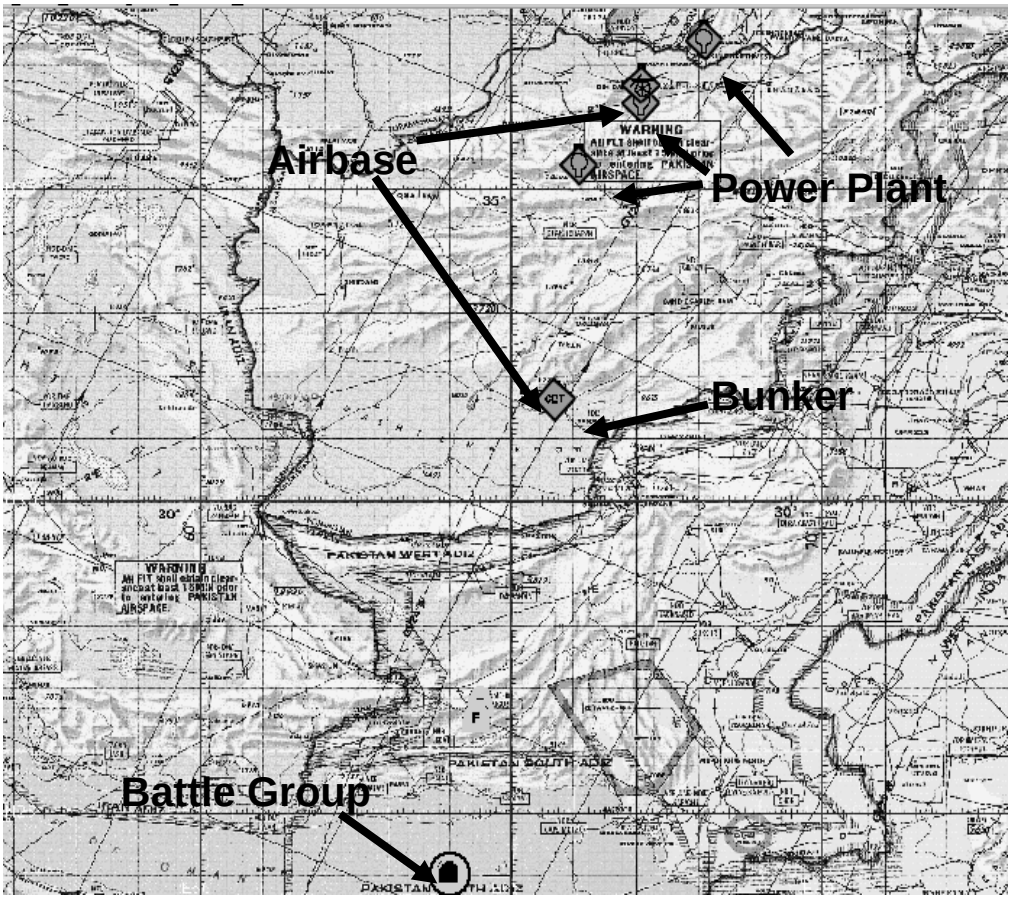

Figure 7. Simple Simulation Scenario

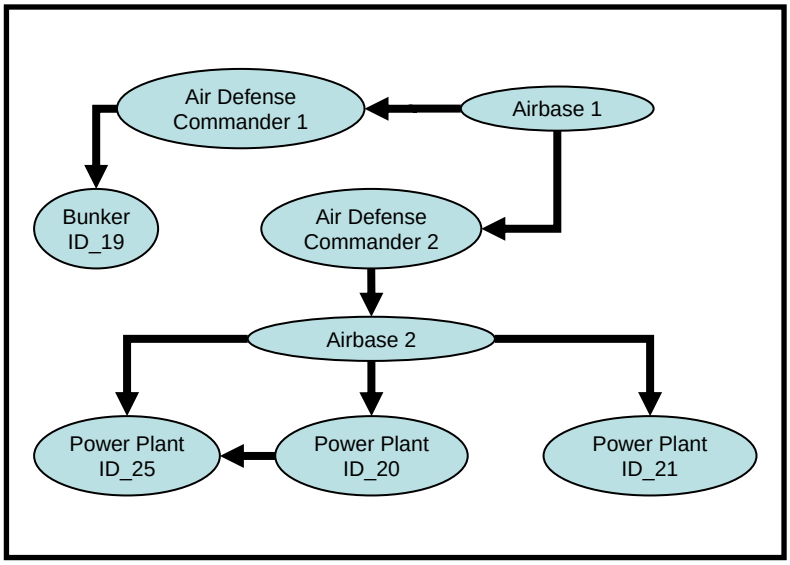

Figure 8. Simple Example of a Center Of Gravity Model

The time line of events can be seen in Figure 9, where the events associated with the COG model are highlighted. In the scenario, four aircraft are launched; they engage the bunker and the three power plants and return to the Battle Group. When simulated without a COG model, the bunker and power plants are destroyed, the SAM is deployed and engages the aircraft, two of which are destroyed. The other two aircraft return safely to the battle group.

When simulated with a COG model employed, the destruction of the bunker causes a cascading event to disable Air Defense Commander 1. The destruction of power plant 25 cascades to disable power plant 20. The destruction of power plant 21, cascades to disable Airbase 2 and its commander, and results in the Airbase 1 being disabled. Because Air Defense Commander 2 was disabled, the SAM was not activated, and all aircraft survive the mission.

Additionally, because the COG model includes recovery times for assets, the power plants, bunker, airbases, and red commanders recover prior to the completion of the simulation. The timing of the recovery is determined by the specific recovery times and dependencies of the COG model. This is shown in Table 1 in the status and readiness columns.

Table 1 compares the end-state of the assets and commanders from this simulation with and without the use of the COG model. The grayed entries highlight that with the use of the COG model, all of the assets are available, and their status is set to 100. Without the use of the COG model, two aircraft are destroyed when the SAM asset is deployed. When the COG model is employed, all aircraft are available, as is the SAM, which was not deployed.

TIME (sec)	Symbol	Event Description	With	Without
625	●	Launch Aircraft	●	●
4625	X	Engage Bunker ID_19;Bunker Destroyed	X	X
4950	⊗	Disable Air Defense Commander 1 Cascading Event	⊗	
6400	X	Engage Power Plant ID_25; Power Plant Destroyed	X⊗	X
6400	⊗	Disable Power Plant 20 Cascading Event		
7050	X	Engage Power Plant ID_21; Power Plant Destroyed	X	X
7200	⊗	Disable Airbase 2; Air Defense Commander 2; Airbase 1 Cascading Event	⊗	
7850	X	Engage Power Plant ID_20; Power Plant Destroyed	X	X
8400	R	Deploy SAM		R
8450	R	Engage Aircraft; Kill 2		R
9150	↑	Recovery of Power Plant ID_25	↑	
9200	↑	Recovery of Airbase 2	↑	
9400	↑	Recovery of Air Defense Commander 2; Airbase 1	↑	
9450	↑	Recovery of Power Plant ID_20	↑	
9600	↑	Recovery of Power Plant ID_21	↑	
11850	↑	Recovery of Bunker ID_19	↑	
12150	↑	Recovery of Air Defense Commander 1	↑	
17500	▲	Land Battle Group; 2 return (with COG four return)	▲	△

Figure 9. Time Line of Events in Simulation

Table 1. Comparison of End State for Simulation (with/without) COG

ID	Asset Name	Status		Readiness	
		no COG	with COG	no COG	with COG
0	Air Defense Commander 1	100	100	available	available
1	Air Defense Commander 2	100	100	available	available
2	Battle Group	100	100	available	available
3	Airbase 1	100	100	available	available
4	Airbase 2	100	100	available	available
5	Power Plant ID_25	0	100	destroyed	available
6	Power Plant ID_21	0	100	destroyed	available
7	Power Plant ID_20	0	100	destroyed	available
8	Bunker ID_19	100	100	available	available
9	Aircraft	43	100	destroyed	available
10	Aircraft	26	100	destroyed	available
11	Aircraft	100	100	available	available
12	Aircraft	100	100	available	available
13	SAM	100	100	deployed	available

3.3 DARPA Coordinators Project

In the final year of the project, the use of FSS to support the evaluation of the Defense Advanced Research Projects Agency (DARPA) Coordinators (Coordination Decision Support Assistants) Program was investigated. The goal of the Coordinators Program is to enable units to adapt mission plans more rapidly, more accurately, with less cognitive load and a greater degree of coordinated action. It will achieve this by creating distributed cognitive systems that automate information exchange and option generation. The challenges include adapting plans in real time by making changes to task timings, task assignments, and by selecting from pre-planned contingencies.

Prior to investigating the use of FSS, the estimation for the task timings was accomplished using a simple random number generation based method. When the simulator received a request to execute a method from an agent, it used a predefined discrete probability function to determine the duration and quality of the performance. The use of the FSS test bed would enable a more realistic estimation of the duration of tasks, and a dynamic discrete event simulation providing information on hypothetical scenarios during the plan adaptation process. However, FSS could not support the scenarios that were required for the evaluation in its present form. In order to provide the required capability, a number of extensions to FSS were required and a SPEEDES external module implemented for interfacing to the C-TAEMs (Coordinators Task Analysis Environment Modeling and Simulation) environment needed to be created. The FSS test bed was modified to enable the following capabilities that are necessary for interactions with Coordinators:

- Add new classes to the FSS software to model tactical teams and their interactions in time and space dependent environmental factors.
- Architect an FSS-Coordinators bridge module, enabling GMASS (Global Infotech Inc. Multi Agent System Simulator) to query FSS for asset and environmental state information.
- Implement/debug architected solution, build test cases.

- Provide the extended FSS software for the evaluation of Coordinators efforts.

Figure 10 depicts the proposed interface between FSS and the Coordinators environment. Two new classes were developed and added to FSS to support this collaboration (Environment and Team). This was done to address the concept of team coordination. If Team 1 has to complete a mission or part of a mission prior to Team 2 performing their mission, but encounters delays due to the environment, Team 2's mission must either be delayed or aborted depending on the scenario/risk, etc. FSS needed to be able to estimate the environmental impact on Team 1's ability to complete their mission and replan Team 2's mission, as necessary. While this was accomplished with extensions to FSS, AFRL's work with the DARPA Coordinators project diminished and the FSS-Coordinators demonstration never materialized. However, due to the research performed in support of the effort, the capabilities of FSS were increased as follows:

- Extended capabilities and links to agent based reasoning engines.
- Extended capabilities to include effects of environmental factors.
- Extended real-time decision support field to include dynamic decisions for blue in addition to emergent adversarial behavior.

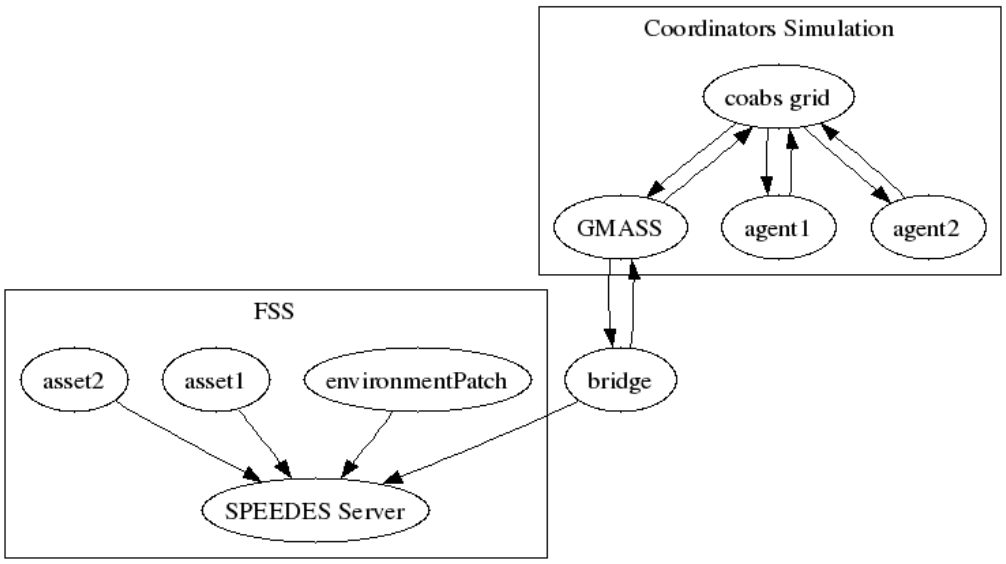

Figure 10. FSS/Coordinators Interaction Environment

3.4 Summary of In-House Activities

Sections 3.1 and 3.2 discussed the core of the in-house research project: creation of FSS, development of an EBO/COG modeling capability, the integration of the EBO/COG modeling capability within FSS and a simple EBO simulation example of its capabilities. Also presented was a short discussion of the simulation capabilities currently resident within FSS. For more details on FSS, the reader is encouraged to refer to Appendix A, the FSS User's Guide. Section 3.3 discussed extensions to FSS based upon potential support for a DARPA project. While this never materialized into a full demonstration, the capabilities of FSS were extended and it also

demonstrated the flexibility of the test bed and showed that it could be easily tailored to other applications.

The following sections will discuss the related research activities that supported our overall research goals; to develop and demonstrate proof of concept technologies to assist decision makers in assessing many friendly effects based COAs against an operational-level adversarial environment, faster than real-time. Readers are encouraged to seek out the references for more details.

3.5 Rapid Decision Branch Analysis

The simulation framework must provide the foundation for rapid decision branch COA analysis. Research was conducted to develop techniques to expand the capability of SPEEDES to evaluate multiple parallel COA simulations, as well as multiple branches, within a single COA. The basic concept is depicted in the lower left of Figure 1. This framework will also provide the capability to input current intelligence, surveillance, and reconnaissance (ISR) reports into a simulation toolset, allowing military planners to rapidly peer into the future at any given moment. Planners are then permitted to derive hypotheses about future alternatives. Notionally, an initial emulation, or basis simulation, could be running in the real-time environment. At critical decision points, the system will clone the simulation and the clone will rapidly execute into the future to evaluate possible outcomes of multiple COAs or decision branches. The cloned simulations leverage the HPC environment by dynamically utilizing free resources for faster than real-time evaluation. By running multiple parallel simulations simultaneously, the maximum computing power can be applied to the problem at hand.

The key to applying simulation technology in real-time is to maintain a simulated “mirror image” of the real world situation at all times. This simulated version of the real world can be used as the starting point for evaluating COAs and for simulating into the future to help predict what might happen. This simulated mirror image is referred to as the emulation. The emulation is a parallel simulation running at wall-clock (i.e. real-time) speed that is continually fed ISR reports so that it reflects the state of the battlespace to the extent known. Assuming that the emulation is run continuously on a defined number of processors of an HPC; the remaining processors will be available to run COA evaluations or predictive simulations.

The approach to creating copies of the emulation for alternative COA analysis is referred to as cloning. Cloning a simulation means creating a running copy of the simulation, preferably on a separate group of “free” CPUs (i.e. processors with little or no workload). For example, if a simulation is running on CPUs 1-8 on a 128-CPU machine, a clone could be created and copied onto CPUs 9-16. The clone will be an exact duplicate of the original and will produce identical results as the original. By cloning the emulation, inserting new COA defining information, and allowing it to run as fast as possible, military analysts can get a glimpse into the possible future, faster than real-time.

This simulation framework provides the foundation for faster than real-time parallel COA simulations. COA analysis is the process of performing “what-if” analysis of actions and reactions designed to visualize the flow of the battle and evaluate each friendly COA. Utilizing the developed methodology, along with performing “what-if” analysis in an HPC environment, affords

the opportunity to evaluate friendly COAs against a range of eCOAs. This range of adversarial COAs goes well beyond the "most likely" and "most dangerous" COAs, which are evaluated in today's current environment. It will encompass a dynamic adversary that responds in an intelligent manner based on friendly actions and adversary models.

Metron, Inc. developed the software framework for building systems to run analytic simulations in real time under an SBIR Phase II project. The framework is referred to as the Dynamic Simulation Framework (DSF). The DSF is a greatly enhanced version of Metron's parallel simulation framework, SPEEDES. The enhancements are designed to allow one to develop systems that run SPEEDES-based simulations in real time on free CPUs in a multiprocessor computing system. The benefits of doing this are:

- Predicting what might happen by simulating quickly into the future.
- Evaluating COAs under consideration at key decision points. This is done by simulating each COA and computing measures of effectiveness (MOEs).

3.6 Automated Scenario Generation

To meet the challenges of future mission planning requirements, a capability was required to be able to generate and assess tens or even hundreds of complete COA scenarios in a matter of minutes or hours. Current scenario generation is extremely labor intensive and often takes several man days to months to develop a limited number of COAs. The process of generating the scenario files is manual and requires a person or persons to collect requisite data from disparate sources and assemble them together into a coherent simulation scenario. This laborious and error prone process presents a significant impediment to the goal of rapidly generating multiple COAs. Furthermore, current scenario generation approaches are data-centric, focusing on the data required by a particular simulation. Data-centric approaches explicitly tie scenario representations to target simulations and data sources, which makes scenario adjustments extremely time consuming. In addition, exercising the same scenarios in multiple simulation environments becomes equally difficult. Current stove-pipe simulations are based on attrition models and are target focused. These simulations are effective in planning for this type of campaign, however, future simulation environments will encompass both effects based and attrition based approaches, and scenario generation must support both environments. This renders the data-centric linkage especially ineffective.

Securboration Inc., under an SBIR Phase II effort, developed an EBO scenario generation (SGen) toolset as an innovative approach to the automated creation of complete scenarios for mission planning simulation. This approach refined how scenario generation technology can be directly applied to problems facing the EBO, Predictive Battlespace Awareness (PBA), mission planning and simulation domains. The SGen toolset breaks the current stovepipe architecture with a robust ontological data model tying mission-planning tools and data resources directly to the Open COA Analysis Framework. As shown in Figure 11, the framework supports multiple target simulations from the SGen Ontology. Securboration's SGen technology breaks the data-centric approach and makes scenario management scenario-centric. Through the use of a campaign ontology, scenario elements and relevant data sources are modeled, making adjustments straight-forward. SGen allows for the rapid development of COAs that can be executed in parallel, faster than real-time.

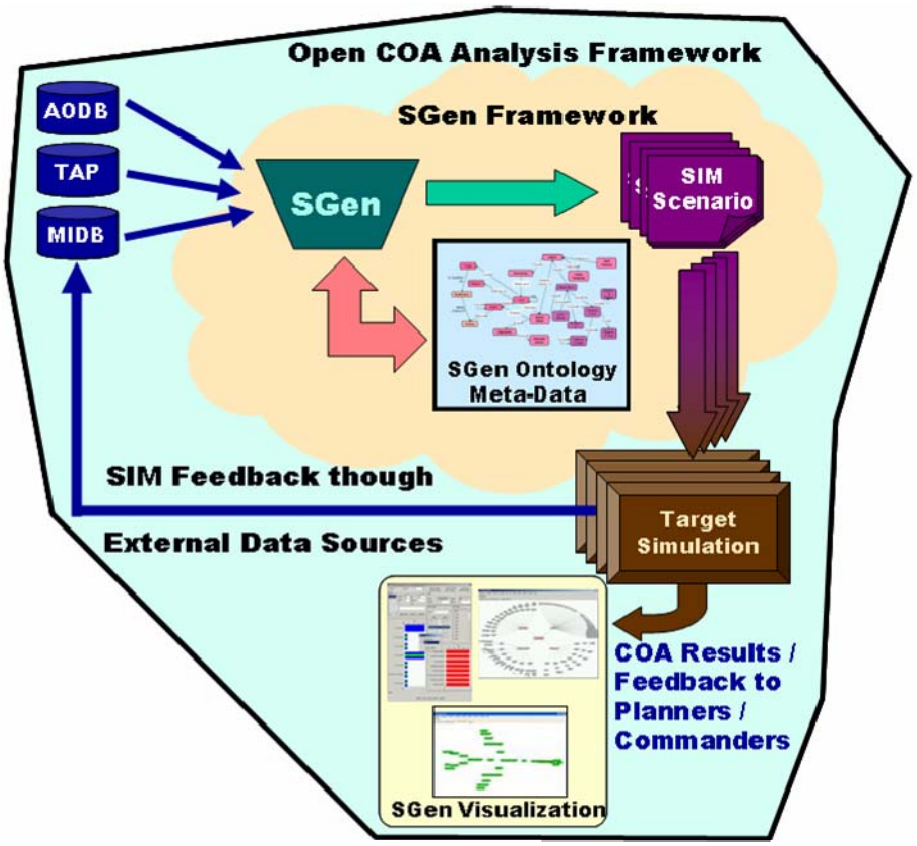

Figure 11. Open COA Analysis Framework

SGen is built around the concept of data integration and analysis through the use of 1) ontological formalization, 2) reasoning and 3) inference capabilities. While most integration and analysis frameworks rely on a model-based approach, or a rule-based approach, SGen uses an ontological-based approach. An ontology-based approach allows for the creation of a meta-model that formally (i.e. supports semantic definitions) represents a domain of interest (e.g. mission planning). This meta-model includes definitions for mapping to domain specific data sources (e.g. Modernized Integrated Database (MIDB), Air Operations Database (AODB) and the AFRL SDT) and definitions for analysis rules (e.g. RuleML). Formalizing the meta-model definition also provides a convenient mechanism for sharing and reusing other ontologies (or models) that have proven to be effective for a particular area of interest. A core set of services were implemented within SGen to insulate the business logic (e.g. scenario allocator) from a specific ontology implementation so that other implementations (e.g. Protégé Knowledge Base) can be used. These services interact through ontology APIs (e.g. Jena and Protégé). Building SGen in this manner allowed it to leverage compatible plug-and-play capabilities of the chosen ontology implementations to perform the functions of declaration, reasoning and inferencing. The SGen ontology was leveraged as part of the AFRL Dynamic Air & Space Effects Based Assessment (DASEA) critical experiment.

3.7 Emergent Adversary Behavior

In the current world environment, the rapidly changing dynamics of adversarial operations are increasing the difficulty for military analysts and planners to accurately predict potential actions.

As an integral part of the planning process, analysts need to be able to assess planning strategies against the range of potential adversarial actions. When the first decision in a given COA is implemented, subsequent decisions must be evaluated based on the new state of the world. This sequential action/reaction analysis concept requires predictive adversary models and these models are vital in assessing planned military decisions. For COA/eCOA analysis tools to be of greater use to military analysts and planners, they must incorporate models of adversarial behaviors that accurately predict potential adversarial actions. Conventional wargaming simulations typically execute a pre-scripted sequence of events for an adversary, independent of the opposing force actions. Traditionally, friendly COAs are wargamed against the "most likely" and "most dangerous" adversary COAs. A significant research challenge for wargaming is predicting and assessing how friendly actions result in adversary behavioral outcomes, and how those behavioral outcomes impact the adversary commander's decisions and future actions.

The feasibility of utilizing an adversarial tool as a core element within a predictive simulation to establish emergent adversarial behavior was investigated. Emergent behavior refers to intelligent dynamic adversarial actions generated at the operational level in response to the execution of the friendly force within the simulation. Multiple adversarial models with varying belief systems would be capable of automatically posing different actions and counteractions. The desire was to use intelligent adversary models to generate alternative futures in performing COA analysis. A significant amount of uncertainty accompanies any adversarial modeling capability. This uncertainty encompasses the process of decision making in a dynamic situation. Typically, models are abstractly created to reflect the adversary's beliefs, goals, and intentions; all of which are based on friendly interpretation of the adversary. The uncertainty of this adversarial decision process makes it necessary to evaluate friendly COAs against a range of eCOAs. Also, based on analysts' interpretations of the adversary, numerous reactions are possible in response to a friendly action. The capturing of these action/reaction dynamics is essential to the future of the COA analysis process. By simulating numerous COAs prior to and during engagement, it may be possible to estimate outcomes of adversary actions immediately after they are accomplished within an operational situation. This will allow decision makers to better respond to a dynamic and volatile adversary during execution, with counter actions.

Two techniques were evaluated for dynamic emergent adversary behavior. In both approaches, adversary actions can be ranked in order of likelihood to occur. By applying HPC technology, it is possible to evaluate multiple adversary actions in parallel, instead of a single decision. Both adversary modeling techniques have been successfully interfaced with FSS. The resultant proof of concept demonstrations showed that it is possible to dynamically execute the adversary force within a simulation. Both techniques are viable alternatives and demonstrate that eCOAs do not have to be pre-scripted. While these approaches were successfully implemented in a proof of concept demonstration, and are continuing to be pursued, they are not the only viable approaches to dynamic adversary modeling. This is a significant research challenge that will continue to be pursued into the future.

3.7.1 Emergent Adversary Modeling System (EAMS)

One technique was investigated under an SBIR Phase II effort by Securboration, Inc. and is called the Emergent Adversary Modeling System (EAMS). It uses an inferencing system framework to infer hypothesis from which dynamic behavior is generated. The inferencing system is based on

Bayesian Knowledge Bases, and captures the overall behavior of the adversary in terms of the following attributes: their general beliefs (about themselves as well as the friendly force), perceptions, biases, and desired end-states. The adversary modeling system inputs observables from a simulation system and infers, in order of likelihood, the adversary goals and intents. From these goals and intents, adversary behavior and resultant actions are calculated. These resultant actions (eCOAs) are then input to the simulation system and executed by the adversary force. The final EAMS deliverable comprised of the following capabilities and approaches:

- Integration with current systems and techniques (e.g. Intelligence Preparation of the Battlefield (IPB)/PBA and FSS).
- The ability to address dynamic behavior, most specifically adversarial (red force) response to the COA the friendly (blue force) forces are executing.
- An architecture that will access knowledge artifacts from multiple sources to create a robust adversarial model as opposed to the flip model often employed (i.e. blue COA is flipped to represent red COA). These knowledge artifacts will be derived from existing systems to assure EAMS interoperability and realism.
- An Adversarial Knowledge Specification Language that will allow the information requirements for adversarial models to be described in a context-free grammar.
- The classification of adversary characteristics, resources and attributes will permit definition of adversarial models in a parameterized fashion. This approach, based on multiple real world adversaries, will support maintainability, extensibility and flexibility in the definition of adversarial models while allowing realistic models to be assembled rapidly.

EAMS has become the foundation component for several ongoing Securboration programs. All of these programs employ an EAMS like inferencing model to provide some form of predictive architecture. Current programs that rely on EAMS technology or use EAMS as a foundation component are as follows: the AFRL DASEA Critical Experiment, an Air Force Office of Scientific Research (AFOSR) funded Small Business Technology Transfer (STTR) Phase II effort titled Dynamic Adversarial Gaming Algorithm (DAGA), and an Office of Naval Research effort titled the Fused Intent System (FIS).

3.7.2 Modeling Adversaries for COA Assessment via Predictive simulation (MADCAP)

The second approach was also investigated under an SBIR Phase II effort by Stottler Henke, Inc. and is called Modeling Adversaries for COA Assessment via Predictive simulation (MADCAP). It utilizes hierarchical planning and ruled-based techniques. The adversary models are based on objectives, actions, predicates and behaviors; and include aspects of cultural and extra-cognitive factors. These attributes are captured by a model execution engine. This execution engine, which was integrated with FSS, is responsible for dynamically determining what actions will be taken by the adversary based on the adversary model and the current state of the simulated world. Some of the key features of the final release of MADCAP include:

- A hybrid agent architecture that enables the creation of adversary agents with different levels of autonomy to maximize the use of computational resources.

- Support for agents that generate plans to achieve their current goals.
- Support for agents that dynamically respond to blue actions, including actions that interfere with the agent's current plans.
- Support for agents that reason about time and space.
- Support for agents that make inferences about the simulated world.
- Support for variation in behavior between adversaries based on differences in personality or culture.
- Support for agents that have imperfect knowledge of the simulated world.

- A generic interface that allows the Adversary Engine to be easily integrated with a wide variety of simulators.
- A domain-independent infrastructure that allows the creation of adversary behavior models in many application areas, both military and non-military.
- A model editor that enables the development of adversary behavior models without writing code.
- A set of visualization tools for viewing and debugging adversary actions during a simulation run.

The MADCAP technology is currently being integrated into Stottler Henke's existing commercial off-the-shelf SimBionic® artificial intelligence middleware package for game and simulation developers. The MADCAP software is also being used in a Phase I SBIR for the Army to control automated role-players in a first-person 3D tactical training simulation.

3.8 Course of Action Simulation Analysis

Simulation of many scenarios for blue against many scenarios for red in an EBO environment poses questions regarding which simulation outcome most closely approaches the intent of the commander. Even in a strictly force-on-force attrition based simulation, the analysis ought to include measures that indicate the relative merit of particular targets and target sets. In an EBO environment, where the targets must map into target sets and centers of gravity, and where indirect consequences to direct actions may in fact be the desired effect, these measures and how to compare simulation results against these measures is problematic.

The CASA effort, which was accomplished by SAIC, researched the underlying technology related to just such an analysis. The primary goal of the CASA project was to create technology that would supply military decision makers with tools to formulate and choose the best COAs available, especially considering the overall goal of the research project, which was the creation and analysis of multiple parallel COAs using HPC. Development and research of scoring procedures has led to the belief that a COA with the highest score may not be the best in terms of the stated goals. An example would be that destroying many enemy targets would not matter if they still had weapons of mass destruction (WMD) capabilities. Therefore, the ability to analyze a COAs data and ensure that the proper metrics are being used is crucial. Different methods to store and display COAs were examined and the ontology data model was chosen. The ability to restructure and manage the data elements and their interrelationships were the reasons to use an ontology data model.

To effectively evaluate a COA, the breakdown of the commander's intent for missions was not detailed enough to produce meaningful scores. Therefore, missions were divided into measures of merit (MOM), measures of effectiveness (MOE), and measures of performance (MOP). These measures let the user inspect missions on the asset level and view individual events, if needed.

In the final year of the CASA effort, the prototype was implemented and the concept of influences was developed. The prototype consisted of three unique COAs that were simulated within FSS and needed to be evaluated. The FSS simulation data were parsed from files and populated into a Protégé ontology. Scores were given to each COA and influences were attached to illustrate their effects. Influences enhanced the ability to model EBO events and were used as checks to make sure conditions in the scenario were met.

4. Conclusion and Recommendations

This report describes the research activities that were pursued over the course of the effort. All activities demonstrated new capabilities and contributed towards the realization of a next generation COA analysis capability; one in which multiple effects based COAs are rapidly developed, analyzed and compared. A major goal of the FSS concept was to develop a test bed to demonstrate associated technologies necessary for performing parallel COA/eCOA simulations, faster than real-time. Developing a complete simulation tool for the end-user community was beyond the research scope. The intent was to demonstrate to user communities the exciting potential for a COA/eCOA technology that will provide increased awareness and insight for decision makers before and during operational situations. Over the course of the project, several technologies were demonstrated related to the research effort.

The initial demonstration, the FSS test bed, was to prove the concept of multiple parallel COA simulations using high performance computers. A generic effects based modeling capability was then developed to allow for the simulation of effects based COAs. This capability was integrated within FSS. An automated scenario generation technique was developed that tied mission data to FSS through an easy to use graphical user interface. This capability leveraged the JavaCOG tool and provided scenarios in minutes/hours verses the traditional days or weeks. The dynamic simulation framework extension to SPEEDES demonstrated the capability of spawning multiple COA simulations from within a single COA simulation for massively parallel COA analysis. This paves the way for multiple "what-if" analysis from within a single COA. Emergent adversarial behavior modeling techniques were developed, and demonstrated the capability of dynamically generating red actions for COA simulations based upon blue actions. This capability will lead to more realistic COA simulations that portray a "red teaming" approach and eliminate the use of static red COAs. Lastly, a capability was developed to rapidly analyze and compare the results of COA simulations. This allows for the quick comparison and ranking of multiple COAs against campaign objectives.

While much was accomplished and many components of the concept were developed and demonstrated during the course of this effort, there is significant work to be accomplished to mature the technologies such that they can be used in a campaign. This effort demonstrated that these automated concepts are possible and can contribute to the future of military planning. While time and manpower didn't allow for further study under this in-house research project, there are numerous research, development and demonstration activities that could be pursued in the future,

based upon what was accomplished and learned. The following is a list, albeit inconclusive, of potential research activities, along with challenges, that should be considered in the future to ultimately achieve massively parallel effects based COA development, analysis and comparison. Also presented are some potential experiments that would help solidify the viability of this approach.

- ***High-fidelity behavior model development***
    - Develop high fidelity models of individuals and groups (adversary as well as allies' and neutral's) which capture intent, motivations, objectives, goals and strategies.
    - Models of how participants will behave under different conditions, a capability to anticipate realistic and unexpected behaviors.
    - Tunable models, depending on the fidelity of the analysis, criticality of the decision, and speed with which a decision must be made.
- ***System of system modeling (SoSM) (sometimes referred to as COG modeling)***
    - Modeling adversary, self and neutrals as complex adaptive systems to synchronize actions with effects; identifying how to stress that system to achieve the campaign goals.
    - Understanding the multidimensional, interdependent nature of the models, strategic to tactical, to support COA development.
    - Developing causal models that link kinetic and non-kinetic actions to direct, indirect, cumulative, cascading and unintended effects taking into account the temporal domain.
- ***COA/scenario generation***
    - Capability to rapidly generate multiple blue COAs (with branches and sequels for foreseeable contingencies) from simultaneous data feeds, both stored and real-time.
    - Ability to tailor the instruments (diplomatic, information, military, economic) of national power to achieve specific effects.
    - Ability to produce COAs for multiple simulation environments.
    - Ability to produce suitable, feasible, acceptable, distinguishable and complete COAs across strategic to tactical dimensions.
    - Accounting for emergent adversary behavior to represent the sequential action/reaction/counteraction nature of operations.
    - Incorporating resources within the generation process.
- ***COA analysis***
    - Simulation of kinetic and non-kinetic actions that achieve direct, indirect, complex, cascading and unintended effects, taking into account the temporal domain.
    - Understanding interdependencies of our systems with the adversary systems.
    - Multi resolution COA modeling & simulation (strategic to tactical).
    - Dynamic exploration of friendly and adversary COAs in the context of one another.
    - Understanding trigger events, what triggers new COAs, spawns COAs and what kills bad COAs.

    - Identifying tipping points, critical decision points, and critical scenario vulnerabilities (in COAs) during analysis.
    - Tunable models, identifying when high fidelity does not equate to better results, identifying the optimum fidelity for the current situation.
- ***COA comparison***
    - Analytical means of comparing dissimilar simulated COAs against predetermined criteria.
    - Metrics and approaches to rank the relative merit of effects based COAs, accounting for causal relationships and dependencies.
    - Visualization of uncertainty/risk/assumptions associated with COAs.
    - Pattern recognition for grouping COAs.
    - Confidence intervals associated with COA analysis results.
    - Identifying COAs that are robust against assumptions.
- ***Managing uncertainty***
    - Propagating/incorporating uncertainty related to data, models, prediction systems etc… utilized throughout the entire planning process.
    - Understanding and visualizing how uncertainty impacts the input/output of decision support systems and its effect on the decision making process.
- ***Design of experiments to identify strengths/weaknesses of tools/techniques***
    - Automated wargaming with emergent adversary behavior vs. blue/red teaming.
    - Automated vs. manual COG analysis.
    - Robustness of COA simulations (variation of results) vs. random number seeds, fidelity of models, numbers of simulations.

The Air Force Research Laboratory is continuing to pursue some of these concepts as part of the Focused Long Term Challenge (FLTC) #1, Anticipatory Command, Control & Intelligence and the AFRL Information and Human Effectiveness Directorate Commander's Predictive Environment (CPE) program.

5. Acknowledgements

The authors would like to acknowledge Mr. Robert G. Hillman for the vision he set forth for this research project and also acknowledge significant contributions made by Mr. Martin J. Walter and Mr. James P. Hanna.

6. References

1. Joint Publication 2-01.3, "Joint Tactics, Techniques, and Procedures for Joint Intelligence Preparation of the Battlespace", May 2000.

2. Fayette, "Effects Based Operations" AFRL Technology Horizons®, vol. 2, no. 2, Jun 2001.

3. McCrabb, "Concept of Operations for Effects-Based Operations", Aerospace Command, Control and Intelligence, Surveillance, and Reconnaissance Center, Langley AFB VA, February 2002.

4. "Synthetic Force Structure Simulation", AFRL-IF-RS-TR-2002-144 Final Technical Report, June 2002.

5. Surman, Hillman, and Santos, "Adversarial Inferencing for Generating Dynamic Adversary Behavior", Proceedings of SPIE: The International Society for Optical Engineering Enabling Technologies for Simulation Science VII: AeroSense 2003, Orlando FL, April 2003.

6. Koziarz, Krause, and Lehman, "Automated Scenario Generation", Proceedings of SPIE: The International Society for Optical Engineering Enabling Technologies for Simulation Science VII: AeroSense 2003, Orlando FL, April 2003.

7. "SPEEDES User Guide", http://www.speedes.com/, April 2003.

8. Joint Publication 3-30, "Command and Control for Joint Air Operations", June 2003.

9. McKeever, Gilmour, and Hillman, "An Approach to Effects Based Modeling for Wargaming", Proceedings of SPIE: International Society for Optical Engineering Enabling Technologies for Simulation Science VIII: DSS 2004, Orlando FL, April 2004.

10. Gilmour, Hanna, and Blank, "Dynamic Resource Allocation in an HPC Environment", Proceedings of the DoD HPCMP Users Group Conference, Williamsburg VA, June 2004.

11. Gilmour, Hanna, Koziarz, McKeever, and Walter, "High-Performance Computing for Command and Control Real-Time Decision Support", AFRL Tech Horizons®, vol. 6, no. 1, February 2005.

12. McKeever, Walter, Gilmour, and Hanna, "Simulating Effects Based Operations", Proceedings of SPIE: International Society for Optical Engineering Enabling Technologies for Simulation Science IX: DSS 2005, Orlando FL, March 2005.

13. Gilmour, Hanna, McKeever, and Walter, "Real-Time Course of Action Analysis", Proceedings of the 10th International Command and Control Research and Technology Symposium, McClean, VA, June 2005.

14. Lehman, Krause, Gilmour, and Santos, “Intent Driven Adversarial Modeling”, Proceedings of the 10th International Command and Control Research and Technology Symposium, McClean, VA, June 2005.

15. Hanna, Reaper, Cox and Walter, “Course of Action Simulation Analysis”, Proceedings of the 10th International Command and Control Research and Technology Symposium, McClean, VA, June 2005.

16. “Effects Based Operations Scenario Generation”, AFRL-IF-RS-2006-128 Final Technical Report, April 2006.

17. McKeever, Gilmour, Lehman, and Stirtzinger, “Scenario Management and Automated Scenario Generation”, Modeling and Simulation for Military Applications, Orlando FL, April 2006.

18. Colenzo, McKeever, DeStefano, and Gilmour, “An Anticipatory Environment Framework”, Proceedings of the 2006 Command and Control Research and Technology Symposium, San Diego, CA, June 2006.

19. Gilmour and Zhang, “Determining Course of Action Alignment with Operational Objectives”, Proceedings of the 2006 Command and Control Research and Technology Symposium, San Diego, CA, June 2006.

20. Gilmour, Krause, Lehman, McKeever, and Stirtzinger, “Scenario Generation to Support Mission Planning”, Proceedings of the 2006 Command and Control Research and Technology Symposium, San Diego, CA, June 2006.

21. “High Performance Computing Engine for Predictive Battlespace Awareness”, AFRL-IF-RS-2006-289 Final Technical Report, September 2006.

22. “Course of Action Simulation Analysis (CASA) Technical Report”, DO 0011 & 0012, submitted to AFRL/IFSD, December 2006.

23. Egan and Reaper, “Course of Action Scoring and Analysis”, Proceedings of the 12th International Command and Control Research and Technology Symposium, Newport, RI, June 2007.

24. “Intelligent, Authorable Adversary Modeling System for Course of Action Assessment”, AFRL-IF-RS-TR-2007-183 Final Technical Report, August 2007.

25. “Campaign Level Adversarial Modeling System”, AFRL-IF-RS-2007-184 Final Technical Report, August 2007.

Appendix A: List of Acronyms

AAA – Anti Aircraft Artillery
AFOSR – Air Force Office of Scientific Research
AFRL – Air Force Research Laboratory
AODB – Air Operations Database
API – Application Programming Interface
C2 – Command & Control
CAP – Combat Air patrol
CASA – Course of Action Simulation Analysis
COA – Course of Action
COG – Center of Gravity
Coordinators – Coordination Decision Support Assistants
CPE – Commander's Predictive Environment
CPU – Central Processing Unit
C-TAEMS – Coordinators Task Analysis Environment Modeling
DAGA – Dynamic Adversarial Gaming Algorithm
DARPA – Defense Advanced Research Projects Agency
DASEA – Dynamic Air & Space Effects Based Assessment
DSF – Dynamic Simulation Framework
EAMS – Emergent Adversary Modeling system
EBO – Effects Based Operations
eCOA – enemy COA
FIS – Fused Intent System
FLTC – Focus Long Term Challenge
FSS – Force Structure Simulation
GMASS – Global Infotech Inc. Multi Agent System Simulator
GPS – Global Positioning System
HLA – High Level Architecture
HPC – High Performance Computing
IPB – Intelligence Preparation of the Battlefield
MADCAP – Modeling Adversaries for COA Assessment via Predictive simulation
MIDB – Modernized Integrated Database
MOE – Measure of Effectiveness
MOM – Measure of Merit
MOP – Measure of Performance
PBA – Predictive Battlespace Awareness
POL – Petroleum, Oil and Lubricants
R&D – Research and Development
SAIC – Science Applications International Corporation
SAM – Surface-to-Air Missile
SBIR – Small Business Innovative Research
SDT – Strategy Development Tool
SEAD – Suppression of Enemy Air Defenses
SGen – Scenario Generation

SoSM – System of System Modeling
SPEEDES – Synchronous Parallel Environment for Emulation and Discrete Event Simulation
STTR – Small Business Technology Transfer
WMD – Weapons of Mass Destruction

Appendix B: User's Guide for the Force Structure Simulator

Table of Contents

Section 1 Introduction

Purpose

This document provides a detailed description of how to use and interact with the Force Structure Simulation test bed (FSS). It should provide the reader with the ability to create the necessary input files for simulating a desired scenario, to execute the simulation, to examine the output, and to interface with the simulation through an external module. The interactions with an external module might be through a map like viewer, an in-line situation monitor module, or a module that drives some part of the simulation (e.g. a red/blue commander).

This document assumes a working knowledge of the SPEEDES Framework and its' API and ready access to the documentation for both, as well as experience with discrete event simulation, such as VHDL or SPEEDES. The SPEEDES Framework is available from www.speedes.com.

FSS Overview

The military planning process, which involves a series of complex decisions in an uncertain environment, is highly manpower intensive and is carried out continuously throughout a campaign. Plans and strategies, which result in courses of action (COAs), are evaluated to determine the necessary steps to meet the overall strategic objectives. Currently, COAs are evaluated by two techniques. One technique involves teams of individuals playing both sides of a campaign, while trying to predict the outcome based on each others actions. This technique is manpower intensive, and cannot be maintained at the speed of current operations. The second technique involves automated wargaming technologies. Automated techniques are faster; however, they are performed against a scripted adversary and focus on attrition based modeling. They are incapable of assessing effects and their contribution to the overall mission objectives, which is inherent in effects based operations (EBO). In an effort to develop technologies to assist decision makers in assessing friendly effects based COAs against an operational-level adversarial environment, faster than real-time a demonstration test bed (FSS) has been implemented. This test bed is built on the SPEEDES Framework which enables the use of many common High Performance Computer architectures including Beowulf clusters.

Installing FSS is discussed in the Installation Guide

FSS supports a force on force simulation which includes cascading and complex effects associated with Effects Based Operations (EBO). Simulation objects include Assets, Commanders, and Center of Gravity objects. Figure 1 shows the hierarchical Class relationships that exist among the various simulation objects. Assets may be fixed or mobile, while Commanders are only mobile. Abstract Centers of Gravity enable the simulation of a work

force, or an abstract attribute like morale. Mobile Assets include SAM assets, Missile assets, Air assets, and Naval assets.

This document discusses the simulation objects, their parameters and attributes in Section 2, examples of using the simulation objects to model specific features in Section 3, the simulation control files in Section 4, sample control files in Section 5, running the simulation in section 6, and driving a simulation from an external module in Section 7.

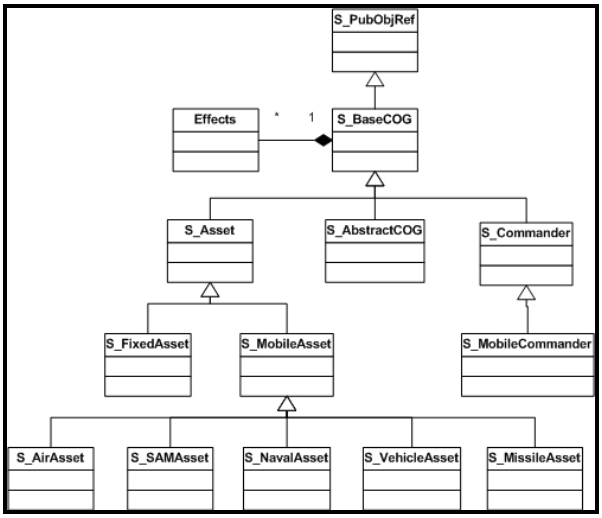

Figure 1. Hierarchy and inheritance for Simulation Objects

Section 2 Simulation Objects

There are three general classes of simulation objects in FSS: Commanders, Assets and abstract centers of gravity (AbstractCOGs). Commanders read the Commanders.par file, and pass Missions, contained in the file to the assets assigned to them. Assets receive Commands from their Commander and execute those Commands. AbstractCOGs subscribe to a set of relational interactions defined in the COG.par file and react to those interactions. Every simulation object has the characteristics 'type' and 'reference' to identify them. Additionally they have an alliance, a readiness, and a status.

Commander objects keep track of the Asset objects that are assigned to them. Assets that report to a Commander are assigned to that Commander. Assets report to their Commander, which can be identified in the Assets.par file, during initialization or as part of a Mission specified in the Commanders.par file.

Every simulation object supports the EBO characteristics of cascading and complex effects, and recovery from those effects. Through the use of the COG.par file (which can be generated with the JavaCOG application), the user can specify the dependency graph, and the interaction relationships including recovery times, and complex effects. Commanders can depend on Assets and/or AbstractCOGs, and so forth through all permutations of Commanders, Assets and AbstractCOGs.

Commander Objects

Every Commander is a MobileCommander, and reads the Missions associated with it in the Commanders.par file. It keeps track of a list Assets it has available and a separate list of the assets it has deployed (given missions to). It has a Plan which holds all the Missions it will assign to its Assets. Commanders have a Name and a Reference that they read from the Commanders.par file. These two characteristics uniquely identify the commander.

During initialization, every MobileAsset reports to its' Commander object, which places information about that asset in its' available assets list. This also occurs at other times which will be described in the section relating to the commands that assets can execute.

When simulation time for a commander reaches a mission time, the commander selects an asset and sends that mission to that asset as a set of commands. The asset's information is moved to the commander's deployed list, and the asset executes the command sequence.

Asset Objects

Assets can be fixed or mobile. Examples of fixed assets are airbases, bunkers, command posts, bridges and power plants. Examples of mobile assets are battle groups, F-16s, SA-2's, tomahawks, and scuds. Only Mobile Assets execute missions.

All assets possess the following characteristics: AssetName, jamming range, radar range, and damage threshold. They may also possess the observability of this asset. Assets may also have a WeaponLoadID. Additionally, mobile assets have an Engagement Range, and a maximum speed.

The characteristic of all Assets are detailed in an Assets.par file, which FSS reads on startup, and uses to populate each Asset object during initialization.

Assets that have commanders specify their CommanderName as part of the Asset.par file. Each Mobile asset reports to its' Commander when the simulation time is 0. This enables each Commander to build its' list of available assets.

An Asset receives a series of commands from its' Commander, and executes those commands at the specified time. By specified time we mean that at the MissionTime, and all of its repetitions the set of commands in the Mission are sent to an Asset one at a time. The Asset begins executing the set when it receives the first command. Commands and their structure and use are described below.

Assets can be dependent on AbstractCOG objects, so abstract things like Morale, and Leadership can disable the ability of an Asset to operate. This is useful for jamming C2 and disabling SAMAssets. Test FSS_020 is an example of the COG.par file defining an AbstractCOG of "C2 operations," and engaging that object from an EC-130 to disable an SA-10.

AbstractCOG Objects

These objects are specified in the COG.par file. Their characteristics include a radar range and a damage threshold. Examples of abstract COGs include Morale, Work Force and other soft attributes associated with the PMESII set of systems.

All simulation objects including Assets, Commanders and abstractCOGs possess COG dependencies; relying on other simulation objects and having other objects depend on them. These dependencies are detailed in the COG.par file.

Using an AbstractCOG, you can model the jamming of the C2 features of a command post or another simulation object. The AbstractcOg entity might be named "C2 Operations", and the simulation object, a CommandPost could be specified in the COGpar file to depend on the AbstractCOG, "C2 Operations." Engaging the "C2 Operations" with a weapon with a COG WeaponClass (say a jamming weapon) can cause the "C2 Operation" to disable, and cascade into the CommandPost, disabling it.

All Assets can depend on AbstractCOGs.

Section 3 Examples of Some Models

The Asset name in the example below is the name of the type of Asset and maps to the ObjType number (401), so that all objects of type 401 in this simulation run will have the AssetName `"Battle Group."` The reference indicates and maps to an initial location that all the assets with that reference will share. For example, any aircraft or missiles associated with that Battle Group will have the reference "USS Enterprise." The Commander identifies the Commander Object that this Asset will report to at the beginning of the simulation. The integer Assets defines the number of these assets that have the same characteristics. If Assets was 2, there would be two Battle Groups with a commander "USS Enterprise Battle Group", and a reference of ""USS Enterprise". This last feature is valuable for mobile assets like SCUDs, SAMs, FA-18s, B-2s, etc.

A simple example of a NavalAsset as a Battle Group:

```
NavalAsset1{
int Assets 1
string AssetName "Battle Group"
int ObjType 401
string Commander "USS Enterprise Battle Group"
string Reference "USS Enterprise"
float LatDegrees 24
float LngDegrees 64
float AltKm 0
string ObjAlliance "Friendly"
float Status 1000
float RadarRange 50
float JammingRange 35
float EngagementRange 10
float DamageThreshold 100
float MaxSpeed 0.309355
float MaxDistance 0
 }
```

LatDegrees, LngDegrees and AltKm are the initial location of the asset. Mobile assets will use this as an initial location; fixed assets use it as an eternal location.

The ObjAlliance specifies whose side this object is on. Allowed values include Friendly, Hostile and Neutral. Take note of the quotation marks.

The Status is the original value assigned to the status attribute. This value is modified during simulation by engagements, and EBO related events. On a recover from an EBO event, the value of status returns to the Status value specified in the Assets.par file.

RadarRange, EngagementRange, JammingRange and MaxDistance are in units of km. The RadarRange is the distance that the object can sense other simulation objects. The EngagementRange is used by SAM Assets to determine at what distance from the SAM it will engage enemy Air and Missile assets. JammingRange is the distance at which a jammer is effective. At present this is scalar, isotropic, and independent of any other attributes of the simulation.

DamageThreshold is the trigger value for status at which the simulation object transitions from operational to destroyed. So, if the Battle Group takes a sufficient number of weapon hits so that its' status drops below 100, the Battle Group will be killed.

Note the WeaponLoadID field in the example below of an FA-18. This refers to an entry in a WeaponLoads.par file, which uses values from a Weapons.par file. If no value is specified or if no WeaponLoads.par file and no Weapons.par file exist in the working directory, the weapontype defaults to maverick, which we've used because it is a precision guided weapon.

Note that Assets is set to 10. In this simulation 10 FA-18s would be assigned to the "USS Enterprise Battle Group" carrying a weaponload of 2. Their Commander, "USS Enterprise Battle Group" will pass missions to them individually, at the start of the mission time. Those missions are described in the Commanders.par file.

A simple example of an AirAsset as a FA-18 is shown below:

```
AirAsset2{
int Assets 10
string AssetName "FA-18"
int ObjType 201
string Commander "USS Enterprise Battle Group"
string Reference "USS Enterprise"
float LatDegrees 24
float LngDegrees 64
float AltKm 0
string ObjAlliance "Friendly"
float Status 100
float RadarRange 30
float JammingRange 20
float EngagementRange 10
float DamageThreshold 50
float MaxSpeed 0.309355
float MaxDistance 0
int WeaponLoadID 2
}
```

An example of a FixedAsset representing an Airport:

```
FixedAsset4{
int Assets 1
string AssetName "Airport"
int ObjType 101
string Commander "Air Defense Commander2"
string Reference "Mazar-e-Sharif"
float LatDegrees 36.7
float LngDegrees 67.1
float AltKm 0
string ObjAlliance "Hostile"
float Status 100
float RadarRange 50
float JammingRange 15
float DamageThreshold 15
 }
```

An example of a FixedAsset representing a Power Plant:

```
FixedAsset8{
int Assets 1
string AssetName "Power Plant"
int ObjType 103
string Reference "ID_20"
float LatDegrees 37.40
float LngDegrees 68.10
float AltKm 0
string ObjAlliance "Hostile"
float Status 100
float RadarRange 0
float JammingRange 0
float DamageThreshold 15
}
```

Note that for the Power Plant, the Reference isn't a name of a city like Mazar-e-Sharif is for the Airport. It refers instead to a geographical location, in this case LatDegrees 37.40, LngDegrees 68.10. Further note that the Power Plant has a radar range of 0, and a jamming range of zero, while those values are non-zero for the airport.

Note that the MaxSpeed in the example below for a SCUD is substantially higher (2.6 vs. 0.3 the units are km/s) than the FA-18, and the EngagementRange is low (10 km), as is the RadarRange (15 km). Though the SCUD won't have radar in reality, or any active sensor, this field is essential to the simulation because it makes it possible for the MissileAsset simulation object to "see" the object(s) it will affect. The SCUD must be able to detect its' target object to schedule an event on it, or to 'engage' the target.

Sample MissileAsset modeling a SCUD:

```
MissileAsset1{
int Assets 12
string AssetName "SCUD"
int ObjType 304
string Commander "Air Defense Commander6"
string Reference "EL POLLO LOCO SCUD BDE HQ"
float LatDegrees 36.3222
float LngDegrees -119.294
float AltKm 0
string ObjAlliance "Hostile"
float Status 100
float RadarRange 15
float JammingRange 0
float EngagementRange 10
float DamageThreshold 10
float MaxSpeed 2.6
float MaxDistance 3000
}
```

Also note that there are 12 SCUDs assigned to the `"Air Defense Commander6",` and at the start of the simulation they will be co-located. For the case where a simulation were to have separate initial locations for each SCUD, with the same Commander, this segment of an Assets.par file would need to be replicated twelve times and modified so that Assets was set to 1, and the lat and lng were set for each individual SCUD.

SCUDs, tomahawks, and seersuckers are all modeled using the MissileAsset even though they are markedly different types of missiles. The use of each missile type would be different in the Commanders.par file. Modeling SCUDs is a little crude. Actual SCUD launchers can be reloaded with another SCUD missile after use. We model the launcher and missile as a single entity. When the missile is launched from the launcher, we model the missile only and the launcher no-longer exists. So, for a SCUD deployment, the missile would move as though it were a ground vehicle. But when it would launch it would move like a TBM. Note that these movements must be specified in the Commanders.par file.

A sample file entry (Assets.par) for a SAMAsset modeling an SA-2:

```
SAMAsset1{
```

```
int Assets 12
string AssetName "SA-2"
int ObjType 505
string Commander "Air Defense Commander5"
string Reference "TWENTY NINE PALMS SA-2 SITE"
float LatDegrees 34.1333
float LngDegrees -115.967
float AltKm 0
string ObjAlliance "Hostile"
float Status 100
float RadarRange 35
float JammingRange 10
float EngagementRange 40
float DamageThreshold 50
float MaxSpeed 0.015955
float MaxDistance 0
}
```

Similar to the SCUD sample all the SA-2s would be co-located at the start of this simulation. Note that the MaxSpeed is substantially less than the FA-18. That is because the SAMAsset models the launcher, a ground vehicle type of movement. SAMAssets are usually autonomous operators. They execute principally via operate and shutdown commands. While operating, the SAMAsset engages every enemy air and missile asset within its' engagement range until it is out of missiles. All SAMAssets, by default, are armed with four missiles, and fire two at a time at a single enemy asset. The use of WeaponLoadID enables other numbers and characteristics of the SAM site to be modeled and simulated. SAMAssets are only able to be engaged by AirAssets when the SAMAsset is operating.

SAM assets also support the 'engage' command, in a GPS like fashion. Given a weapon designation that corresponds to a GPS class weapon, and a route (a location in space), the SAM engages Air and Missile Assets within the HitRadius of the location and the engagement range of the SAMAsset. AAA can be modeled as a SAMAsset with a WeaponLoad containing many GPS like weapons, and receiving repetitive engage commands. An Example of a SAMAsset used as AAA is contained in test FSS_019.

Section 4 Simulation Control Files

There are several Simulation Control Files. We've mentioned previously the Objects.par file. There are several other par files that are SPEEDES related and the user should refer to the Speedes User's Guide for those files, including the InterestSpaces.par file, and the speedes.par file.

This section discusses the simulation control files: Assets.par; Commanders.par; COG.par, Weapons.par; and WeaponLoads.par. It describes the contents of each file as well as its' use. Examples of the simulation control files are in the FSS/Tests directory. This directory is the regression test directory. Each subdirectory (FSS_* and FM_*) has a set of par files to be executed to test FSS for some capability. A REAME file in each directory describes the scenario being tested, and specifies the results of the sim. The Assets.par, Weapons.par,

WeaponLoads.par COG.par and Commanders.par files are a good source of examples to copy from. Tests with numbers above 017 test for some of the more recent features: CAP, SEAD, interdict, GPS in SAM assets, Teams, deploy and recover commands.

Assets.par File

Examples of segments of the Assets.par file were shown in the preceding section of this document. The file is composed of a collection of SpSets: `AirAsset, NavalAsset`, `SAMAsset`, `MissileAsset`, `VehicleAsset`, Team, and `FixedAsset.`

These SpSets may exist in the Assets.par file even if they are empty. A very simple and useless example of an Assets.par file would be:

```
AirAsset{}
NavalAsset{}
SAMAsset{}
MissileAsset{}
VehicleAsset{}
Team{}
FixedAsset{}
```

Note that the '{}' pairs define the contents of the SpSet. For the SA-2s shown earlier the SAMAsset SpSet would be

```
SAMAsset{
    SAMAsset1{
      int Assets 12
      string AssetName "SA-2"
      int ObjType 505
      string Commander "Air Defense Commander5"
      string Reference "TWENTY NINE PALMS SA-2 SITE"
      float LatDegrees 34.1333
      float LngDegrees -115.967
      float AltKm 0
      string ObjAlliance "Hostile"
      float Status 100
      float RadarRange 35
      float JammingRange 10
      float EngagementRange 40
      float DamageThreshold 50
      float MaxSpeed 0.015955
      float MaxDistance 0
    }
}
```

Commanders.par file

This file is a collection of SpSets. The Set called Objects references the Sets for each Commander.

```
Objects{
  reference Commander0  // Air Defense Commander2
  reference Commander1  // Air Defense Commander3
  reference Commander2  // Air Defense Commander4
  reference Commander3  // Air Defense Commander4
  reference Commander4  // Air Defense Commander5
  reference Commander5  // Air Defense Commander5
  reference Commander6  // Air Defense Commander6
  reference Commander7  // Air Defense Commander7
```

```
  reference Commander8  // AirForce Commander
  reference Commander9  // USS TR Group Commander
}
```

Each Commander SpSet has at least two fields “Commander” and "Reference." For example:

```
Commander0{
   string Commander "Air Defense Commander2"
   string Reference "Mazar-e-Sharif"
}
```

Commander is the name of the Commander and is the same as the ‘Commander’ field in the SA-2 example in the Assets.par file above. Each Commander SpSet also contains zero or more SpSets for Missions.

Mission SpSets have an Asset name that will receive the Mission, a Mission time, a set of Commands, which is another SpSet, and a few optional fields such as WeaponLoad and asset id numbers. These latter optional fields enable the ability to specify which specific asset to assign the Mission to, and what weaponloadID the asset that receives the Mission must have in order to execute the Mission. This means that one could have tomahawks with differing weaponloads and give missions to the tomahawks based on the type of ordinance on-board. Setting the Asset Ids amounts to listing the simulation global ID (or IDs) for the objects that will execute this mission.

Sample Mission for a Gadfly (a type of a Surface to Air Missile Asset; SAMAsset).

```
Mission1{
string AssetName "Gadfly"
string MissionTime "2:20"
   commands{
      Command001{
        string CommandName "move"
        float Speed 20
        string route "route44"
      }
      Command002{
        string CommandName "operate"
      }
      Command003{
        string CommandName "report"
        string CommanderName "Air Defense Commander2"
      }
   }
 }
```

In this example, a single Gadfly receives the Mission when the simulation time is at 2 hrs and 20 minutes (8400 seconds). It receives a command to move at a speed of 20km/hr along route44, which is specified later in the Commanders.par file. When the move command is completed, the Gadfly goes into the operation mode, and reports to the commander. The specifics of Commands are discussed later in this text.
This Gadfly example used the line:

```
string MissionTime "2:20"
```

This indicates a single time, when a single Asset executes the mission. Alternatively, a set of sorties can be specified as:

```
string MissionTime "{3,  0:20,  1}"
```

In this example, three assets are deployed, the first one when GVT is 20 minutes, and each successive one at 1 minute intervals. Each asset receives an identical mission.

The optional fields for a mission SpSet:

```
int NewMission 1
int WeaponLoad 10
int AssetIDs 22 45 37 12
```

NewMission may have the value 0, 1, or 2. In a Commanders.par file only 1 and 2 make any sense. 1, which is the default if the field is omitted indicates that the mission should start. 2 indicates that that this mission retasks the asset and aborts any existing mission.

The value of WeaponLoad is the ID of the weapon loads found in WeaponLoads.par That file is discussed in detail later. If WeaponLoad is omitted the simulation defaults to WeaponLoad 1, which is 6 AGM-65s.

AssetIDs is a space separated list of the simulation global ids of the assets that will receive this mission. If the MissionTime indicates that there will be three sorties, and AssetIDs has less than 3 ids of the correct type, the simulation aborts. The use of AssetIDs is particularly useful for cases of driving the simulation from an external module, and a specific asset is supposed to be retasked or tasked with a mission.

In addition to fields previously mentioned, each Mission SpSet has an SpSet named commands. The commands SpSet is a set of Command SpSets each of which can have several fields. Each Command SpSet has a CommandName. The other fields which are used based upon the type of command are: time, Radius, Speed, route, Target, Reference, WeaponName, WeaponDesignation, and SubSystemName.

The Motion Command

There are three types of Commands: MotionCommand, InteractionCommand, and StateCommand. MotionCommands are directions to execute some kind of movement. That movement can be either a GreatCircle movement or a loiter movement. To execute a great circle movement the Command SpSet would be:

```
Command007{
   string CommandName "move"
   float Speed 700
   string route "route35"
}
```

The combination of fields Speed, route and the CommandName “move” indicates a Great Circle Movement from the current position of the MobileAsset along route35 at a constant speed of 700 km/hr.

Loiter Motion is achieved by specifying the fields: time, Speed, route and Radius.

For example:

```
Command006{
    string CommandName "move"
    string time "1:00"
    float Speed 400
    string route "route20"
    float Radius 5
}
```

Speed is the speed of the Asset as it traverses its’ circular course, time is the duration that the asset loiters, Radius is the radius of the circle about the location that the asset moves (in km) , and route is the location. Though this is called route, and references a route SpSet, the route should hold a single Vector SpSet.

The MotionCommand structure supports a Rendezvous motion, though the Assets do not as yet support the execution of such a command. The structure for a Rendezvous motion requires the following fields: time, and route. It may not possess a Speed or a Radius(e.g. no Speed and no Radius mean this is a Rendezvous motion). Under a Rendezvous command, an asset would use a speed required to place the Asset at the end of the route at time ‘time.’

Interaction Commands

InteractionCommands direct an Asset to interact in some way with another simulation object. These commands include: engage, land, report and assume. Only AirAssets support the land command. A sample land command is shown below:

```
Command004{
   string CommandName "land"
   string Target "Battle Group"
   string Reference "USS Roosevelt"
}
```

This command instructs an AirAsset to land on or at another Asset. The Target field is the AssetName, in this case ‘Battle Group,’ while the Reference field is the corresponding reference for that simulation object. In this case the USS Roosevelt Battle Group.

Assume and report take the form:

```
Command005{
   string CommandName "assume"
   string CommanderName "USS Enterprise Battle Group"
}
```

The command ‘report’ notifies a commander that this asset is available to receive a mission. The command ‘assume’ notifies the commander that the asset is the home of the commander.

The ‘engage’ command is a little complicated. The simulation supports several classes of weapons. Each class requires a specific set of fields for the engage command. Weapon Classes include:

ActiveSensor
GPS
WMD
COG

The first class is Active Sensor weapons, such as a maverick or HARM, which employ the Target and Reference fields to specify a particular target. So, one might change CommandName in Command004 above to “engage” and the Asset would engage the Battle Group. Note that killing the Battle Group does not destroy the assets that share the same reference, say for example the FA-18s on board. This is a limitation of the test bed at this time.

InteractionCommands may have the CommanderName field or the Target field, but not both. Engage commands can use either the WeaponName or WeaponDesignation field to specify which Weapon to use to engage this target. This supports heterogeneous WeaponLoads, i.e. a mix of types of weapons on the Asset (maverick, gbu-15s, mk-82s). The WeaponName and WeaponDesignation fields correspond to items in the Weapons.par file. Lack of a WeaponName or WeaponDesignation causes the simulation to default to maverick. Specifying a WeaponDesignation, or WeaponName that does not exists in the simulation results in segmentation fault, with an error message. Specifying a WeaponDesignation that exists in the simulation, but does not reside on the Asset, results in failed engagements and a warning message that no weapon of that type was found.

A ‘route’ field may appear as part of the engage command in order to support GPS type weapons. Like the Loiter motion, a single Vector should exist in the route.

Sample engage commands:

ActiveSensor-like

```
Command006{
  string CommandName "engage"
  string Target "Power Plant"
  string Reference "ID_20"
}
```

GPS like engagement

```
Command007{
  string CommandName "engage"
  string route "route1"
  string WeaponDesignation "GBU-15"
}
```

The ‘Target’ field need not specify an Asset. It might specify an abstractCOG element like workforce:

```
Command008{
  string CommandName "engage"
  string Target "Work Force"
  string Reference "ID_20"
  string WeaponDesignation "PW-100"  (leaflets)
}
```

State Commands

StateCommands include ‘operate’, ‘shutdown’, ‘wait’, ‘CAP,’ ‘SEAD,’ ‘AntiSCUD,’ ‘interdict’, and ‘MC’ (for mission complete). These commands each have a time field. The time field is the time that the command takes to execute, or the time until the next command will execute (in simulation time). In AirAssets, the use of the ‘time’ field is questionable, because the asset remains at its current location until the StateCommand completes.

So, if an AirAsset executes a move, operate jammer, move sequence, the asset does not begin the second move command until the ‘operate jammer’ command’s time is completed. If one were to desire an AirAsset to execute a loitering motion while operating in CAP mode, the sequence of commands would be:

1. move to the area
2. CAP
3. move in a loiter motion

The use of a time field in a StateCommand would leave the air asset hanging in the air for that period of time.

The commands ‘operate’ and ‘shutdown’ optionally can include a SubSystemName field for turning on/off a subsystem, such as a jammer. They default to complete system for use with ‘operate’, ‘CAP’, ‘SEAD’, ‘interdict’ and ‘AntiSCUD.’ ‘shutdown’ is the counter command for ‘operate’, ‘CAP’, ‘SEAD’, ‘interdict’ and ‘AntiSCUD.’

Example:

```
Command009{
  string CommandName "operate"
  string SubSystemName "jammer"
}
```

OR

```
Command009{
  string CommandName "CAP"
}
```

‘CAP’ which stands for Combat Air Patrol, ‘SEAD’ which stands for Suppression of Enemy Air Defenses, ‘interdict’ which indicates engagement against all enemy Assets, and ‘AntiSCUD’ (self explanatory) are automatic engage commands. They change the state of a MobileAsset, so that it uses a sensor scan to find enemy Assets of a particular class (CAP: AirAssets; SEAD: SAMAssets; AntiSCUD: MissileAssets; interdict: ALL enemy mobile Assets) and engages them whenever they are within the EngagementRange of that MobileAsset. Assets can receive and execute particular engage commands (as described earlier in this document), while they are in an automatic engage state.

The use of EngagementRange has a far reaching effect on the simulation. Operating in SEAD, an AirAsset engages any and all SAMAssets within its engagement range. However, the default values for EngagementRange in the regression tests have larger values for most SAMAssets than for the AirAssets.

Air-to-Air engagements occur in CAP, and the outcome depends not only on EngagementRange, but additionally on the timing of the engagements, and of course the Pk of the weapons employed.

CAP and SEAD do not support any weapon designation, and the assets just pull whatever weapons are on their WeaponLoad, one at a time until it is exhausted. This requires planning on the part of the creator of the engagements. Without careful planning, you might try to engage an AirAsset with a Maverick or with Leaflets.

An example of SEAD and CAP usage has been included in the regression test suite as FSS_017, interdiction is found in test FSS_025.

Following the Commander sections of the Commanders.par file are a set SpSets that hold the routes.
These SpSets are stand alone sets. They each consist of a number of SpSets labeled Vectors that have elements of LatDegrees, LngDegrees, and AltKm. As is indicated by the name, LatDegrees and LngDegrees have units of degrees, while AltKm is in units of km above mean sea level.
Example route SpSet:

```
route44{
// SAM Deployment
    Vectors0{
      float       LatDegrees  36.69
      float       LngDegrees  67.11
      float       AltKm  0.0
    }
 }

route45{
// Qarshi to  Mazar-e-Sharif
    Vectors0{
```

```
      float       LatDegrees  38.0
      float       LngDegrees  67.0
      float       AltKm  10.0
    }



    Vectors1{
      float       LatDegrees  36.7
      float       LngDegrees  67.1
      float       AltKm  5
    }
}
```

An example of a simple route is ‘route44’, which has a single Vector to indicate that the endpoint for the motion is this location. The simulation crashes if a mobile asset attempts to move to the place that it already exists (there’s a divide by zero error). Multipoint routes are specified as shown in route45.

At the end of the Commanders.par file there is an SpSet labeled RegionOfInterest. This set holds the TheaterName, which must be identical to the Space reference in the InterestSpaces.par file.

Example of RegionOfInterest SpSet:

```
RegionOfInterest{
    string TheaterName "SW_AsiaTheater"
}
```

Example Commanders.par file Segment

This section details an example of how one might simulate a SCUD. This file segment is for the SCUD Commander. The Commander sends the SCUD, whose Global ID is 22 on a ground track when GVT is 1:20. The Commander moves it again on a ground track when GVT is 5:00. Then, when GVT is 26:00, it launches the SCUD to engage the AirBase at Nellis.

```
Commander0{
   string Commander "TBM Commander"
   string Reference "UI SCUD HQ"
    Mission1{
    string AssetName "SCUD"
    string MissionTime "01:20"
    int AssetID 22
       commands{
          Command001{
            string CommandName "move"
            float Speed 20
            string route "route44"
          }
       }
    }
    Mission2{
    string AssetName "SCUD"
    string MissionTime "5:00"
    int AssetID 22
```

```
      commands{
         Command001{
           string CommandName "move"
           float Speed 20
           string route "route50"
         }
      }
   }
   Mission3{
   string AssetName "SCUD"
   string MissionTime "26:00"
   int AssetID 22
      commands{
         Command001{
           string CommandName "move"
           float Speed 2500
           string route "route005"
         }
         Command002{
           string CommandName "engage"
           string Target "AirBase"
           string Reference "Nellis AFB"
         }
      }
   }
}
```

COG.par File

This parameter file is optional. The file is necessary for the simulation to run with EBO capabilities (Cascading, Complex, Recovery events), if the file is absent from the working directory FSS will run in attrition mode. It is also necessary if the simulation requires abstract COG elements.

The COG.par file is a collection of SpSets. This file has two sections; one that describes the interrelations between simulation objects and the other provides the information for the abstractCOGs.

AbstractCOGs

If the simulation requires abstract COGs then there is a SpSet titled “AbstractCog.” An example is listed below. If the Simulation does not require Abstract COGs then this SpSet can be omitted from the parameter file.

```
AbstractCog{
   AbstractCog1{
    int   Assets          1
    string       AssetName       "Work Force Morale"
    int   ObjType         123
    string       Reference       "ID_21"
    string       ObjAlliance     "Hostile"
    float        Status          100
    float        RadarRange      0
    float        DamageThreshold 50
   }
}
```

The AbstractCog SpSet is composed of inner SpSets. Each of these subsets represents an abstract COG entity. All of the attributes listed within the subsets are the same as other assets listed in the Assets.par file. This was done to provide the abstract COG objects with as much flexibility as possible.

EBO Properties

The next core section of the COG.par file is where the EBO and dependency properties are listed. This section also transforms the simulation from an attrition-based to an effects-based simulation.
A small section is described below.

```
Objects{
 reference ebo0  // Airport ID_1
 reference ebo1  // Airport ID_2
 reference ebo2  // Power Plant ID_25
 reference ebo3  // Power Plant ID_20
}

Ebo0 {
   string  name "Airport ID_1"
   double timeDown 1280.0
   double percentageThreshold 50.0
   effectedSets{
     Set0{
        string effectedBy "Power Plant ID_21"
        double probability 1.0
        double restore  120.0
        double delay 150.0
     }
     Set1{
        string effectedBy "Power Plant ID_20"
        double probability 1.0
        double restore  120.0
        double delay 100.0
     }
     Set2{
        string effectedBy "Power Plant ID_25"
        double probability 1.0
        double restore  120.0
        double delay 75.0
     }
  }
}
ebo1{
string name "Airport ID_2"
float timeDown 128000
string windicator "Air Traffic in Area ID_2 is normal"
string findicator "Air Traffic in Area ID_2 is reduced by 95%"
   effectedSets{
      set0{
         string effectedBy "Power Plant ID_21"
         double probability 1
         double delay 100.0
         double restore 10.0
         string complexEffect "Power Plant ID_25" "Power Plant ID_20"
      }
      set1{
         string effectedBy "Power Plant ID_25"
```

```
         double probability 1
         double delay 100.0
         double restore 0.0
         string complexEffect "Power Plant ID_20" “Power Plant ID_21"
      }
      set2{
         string effectedBy "Power Plant ID_20"
         double probability 1
         double delay 0.0
         double restore 0.0
         string complexEffect "Power Plant ID_21" "Power Plant ID_25"
      }
   }
}
```

Objects is a SpSet of that Contains the simulation entities’ EBO properties. Each entity is comprised of four attributes and a SpSet. The first attribute is “name” which is a string that uniquely identifies the current object. This name is comprised of the simulation object’s “name” and “Reference.”

Next attribute is “timeDown” which is the time that the entity requires to recover from a direct effect. This time is measured in seconds.

The next two attributes are windicator and findicator. These attributes are the indicators for the current entity. Windicator is the working indicator and findicator is the failure indicator. Both of these attributes are strings.

An optional attribute percentageThreshold is a double and determines how this object will calculate its status. If this attribute is present, the object requires the given percentage of its children (defined in EffectedSets) to be operating to remain operational. In the example above “Airport ID_1” requires fifty percent of its three children to remain operational. When the percentageThreshold is not present the object calculates its status using the default method, which is defined next.

Next is the SpSet, effectedSets. EffectedSets is comprised of subsets that describe the EBO properties and dependency relationship between the parent node (ebo# set) and the current set (set#). In the example above, ebo1, the asset “Airport ID_2” relies on three objects; “Power Plant ID_20”, “Power Plant ID_21”, “Power Plant ID_25”. Each of the subsets in the EffectedSets consists of five attributes.

The first property defined is the “effectedBy” attribute, this gives the name of the entity that this object depends on. This name is the concatenation of the object’s name and reference.
The next attribute is “probability”, this is the likelihood that the entity named in this set will affect the object named in the ebo set. The probability has a range of 0 to 1, with 1 meaning it will certainly disable the object. When a simulation is run with a seed value (see section 6), the probability values will become more random. A probability of 1 will become a random number between 0.81 and 1, a probability of 0.5 will be a value between 0.4 and 0.6, and probability of 0 will be a value between 0 and 0.2.

The “delay” attribute allows users to set how long after the entity listed in the set has been disabled, that the object will calculate the effect. When an entity is disabled or destroyed this object will not be effected by that entity until the “delay” time is reached after the entity was affected.

The restore attribute gives the user the ability to set a delay time in a recovery event. So when the entity in the set recovers, the object will wait the given time to recover from that cascading effect.

The last attribute, complexEffect, allows users to define complex relationships between assets. This attribute is an array of strings. For the object to be disabled by this entity (effectedBy), then all the entities listed after complexEffect will had to been disabled or destroyed.

In the example above if Power Plant ID 21 is destroyed or disabled then it will have an indirect effect on Airport ID_2. There is a very high probability of this effect disabling Airport ID_2 in 100 seconds. However Airport ID_2’s complex relationship is defined that if Power Plant ID_21 is destroyed or disabled it will only have a disabling effect on Airport ID_2 if Power Plant ID_20 and Power Plant ID_25 are also destroyed or disabled.

Weapons.par File

This file is an SpSet. The Set called Weapon contains all the SpSets for Weapons that can be used in the simulation. Each Weapon subset (an SpSet) has seven fields. Example of the default Weapon:

```
Weapon1{
    int WeaponType 1
    string WeaponName "Maverick"
    string Designation "AGM-65"
    int WeaponClass 100
    float Range 27
    float P_kill 0.9
    float HitRadius 0.1
}
```

The ‘WeaponType’ field is unique for each type of weapon, all mavericks in the simulation will have the same ‘WeaponType’, 1, and the same ‘WeaponClass’, 100. The ‘WeaponClass’ field allows the simulation to support GPS guided, Active Sensor guided, and WMD type weapons. The WeaponName and WeaponDesignation are two ways to refer to this type of weapon in the Commanders.par file. The ‘Range’, ‘P_kill’, and ‘HitRadius’ are fields that specify characteristics of this type of weapon. P_kill is the probability of affecting the target; the range is the distance in km that the weapon can be employed, and the ‘HitRadius’ is used with GPS guided weapons to define the range in which a GPS guided Missile Asset will look for a target to engage. Additionally, the WMD class of weapons supports the field ‘KillRadius’. The WMD class of weapons use the kill radius as the range over which they effect assets. A weapon with a kill radius of 5 km schedules an event on every asset whose distance to the weapon is less than or equal to 5 km.

WeaponClass is an integer that maps through the WeaponClassEnum:
ActiveSensor(AS) = 100, GPS guided (GPS) = 101, INS = 102,
WeaponofMassDestruction(WMD) = 103, Abstract COG (COG) = 104.

The Class mapping is defined in the code in ActionTypeEnum.h

WeaponLoads.par File

This file is also an SpSet. The Set called WeaponLoads contains every SpSet defining a WeaponLoad that can be used in the simulation. Each WeaponLoad subset (an SpSet) has two fields. The first is the ID, which uniquely identifies the WeaponLoad for the simulation and is used by the Commanders.par and Assets.par files. The second field is an SpSet called Weapons which is composed of one or more SpSets for each WeaponType in the WeaponLoad. Each of these SpSets has two fields, a Designation which maps to the WeaponDesignation in the Weapons.par file, and a Number which indicates how many of this type of weapon are in the WeaponLoad. WeaponLoad1 is the default configuration if no WeaponLoads are specified in the simulation control files. Weapons.par and WeaponLoads.par are not required to execute the simulation. However, the Assets.par file and Commanders.par file must not reference WeaponLoads or Weapons if they are omitted.

An example of a WeaponLoads.par file follows:

```
WeaponLoads{
    WeaponLoad1{
        int ID 1
           Weapons{
              Weapon001{
                string Designation "AGM-65"
                int Number  6
              }
           }
     }
    WeaponLoad2{
        int ID 2
           Weapons{
              Weapon001{
                string Designation "AGM-65"
                int Number  2
              }
              Weapon002{
                string Designation "MK-84"
                int Number  4
              }
           }
     }
}
```

Section 5 Sample Simulation Control Files

This section contains a set of control files for a sample simulation. It employs COGs, leaflets and mavericks from AirAssets (that are launched from a NavalAsset) and a FixedAsset. First we show the 'Weapons.par' and 'WeaponLoads.par' files.

Sample Weapons.par file

```
Weapon{
  Weapon1{
      int WeaponType 1
      string WeaponName "Maverick"
      string Designation "AGM-65"
      int WeaponClass 100
      float Range 27
      float P_kill 0.9
      float HitRadius 0.1
  }
  Weapon2{
      int WeaponType 2
      string WeaponName "DUMB Bomb"
      string Designation "MK-84"
      int WeaponClass 100
      float Range 10
      float P_kill 0.6
      float HitRadius 0.013
  }
  Weapon3{
      int WeaponType 3
      string WeaponName "leaflets"
      string Designation "PW-001"
      int WeaponClass 104
      float Range 10
      float P_kill 0.7
      float HitRadius 0.31
  }
}
```

Sample WeaponsLoad.par file

The WeaponLoads.par file makes reference to the WeaponType, WeaponName or WeaponDesignation from the Weapons.par file.

```
WeaponLoads{
   WeaponLoad1{
      int ID 1
      Weapons{
         Weapon001{
            string Designation "AGM-65"
            int Number  6
         }
      }
   }
   WeaponLoad2{
      int ID 2
      Weapons{
         Weapon001{
            string Designation "AGM-65"
            int Number  2
         }
         Weapon002{
            string Designation "MK-84"
            int Number  4
```

```
            }
         }
      }
      WeaponLoad3{
         int ID 3
         Weapons{
            Weapon001{
               string Designation "PW-001"
               int Number  2
            }
         }
      }
   }
```

Sample COG.par file

```
   AbstractCog{
       AbstractCog1{
           int Assets 1
           string AssetName "Work Force"
           int ObjType 123
           string Reference "ID_21"
           string ObjAlliance "Hostile"
           float Status 100
           float RadarRange 0
           float DamageThreshold 50
       }
       AbstractCog2{
           int Assets 1
           string AssetName "Work Force"
           int ObjType 123
           string Reference "ID_20"
           string ObjAlliance "Hostile"
           float Status 100
           float RadarRange 0
           float DamageThreshold 50
       }
           AbstractCog3{
           int Assets 1
           string AssetName "Work Force"
           int ObjType 123
           string Reference "ID_256"
           string ObjAlliance "Hostile"
           float Status 100
           float RadarRange 0
           float DamageThreshold 50
       }
   }
```

```
Objects{
 reference ebo0  // Airport Kandahar
 reference ebo1  // Power Plant ID_21
 reference ebo2  // Power Plant ID_20
 reference ebo3  // Power Plant ID_25
 reference ebo4  // Airport Mazar-e-Sharif
 reference ebo5  // Air Defense Commander2 Mazar-e-Sharif
 reference ebo6  // Airport Herat
 reference ebo7  // Work Force ID_20
}

ebo0{
string name "Airport Kandahar"
float timeDown 500
string windicator "none"
```

```
string findicator "none"
   effectedSets{
      set0{
         string effectedBy "Airport Herat"
         double probability 1
         double delay 0.0
         double restore 10.0
         string complexEffect "Bunker ID_19"
      }
      set1{
         string effectedBy "Bunker ID_19"
         double probability 1
         double delay 60.0
         double restore 0.0
         string complexEffect "Airport Herat"
      }
   }
}

ebo1{
string name "Power Plant ID_21"
float timeDown 2550
string windicator "none"
string findicator "none"
   effectedSets{
      set0{
         string effectedBy "Work Force ID_21"
         double probability 1
         double delay 100.0
         double restore 0.0
      }
   }
}

ebo2{
string name "Power Plant ID_20"
float timeDown 4350
string windicator "none"
string findicator "none"
   effectedSets{
      set0{
         string effectedBy "Work Force ID_20"
         double probability 1
         double delay 100.0
         double restore 0.0
      }
   }
}

ebo3{
string name "Power Plant ID_25"
float timeDown 2750
string windicator "none"
string findicator "none"
   effectedSets{
      set0{
         string effectedBy "Work Force ID_256"
         double probability 1
         double delay 100.0
         double restore 0.0
      }
   }
}

ebo4{
string name "Airport Mazar-e-Sharif"
float timeDown 1280
```

```
string windicator "none"
string findicator "none"
   effectedSets{
      set0{
         string effectedBy "Power Plant ID_21"
         double probability 1
         double delay 150.0
         double restore 10.0
         string complexEffect "Power Plant ID_25" "Power Plant ID_20
      }
      set1{
         string effectedBy "Power Plant ID_25"
         double probability 1
         double delay 100.0
         double restore 0.0
         string complexEffect "Power Plant ID_20" "Power Plant ID_25"
      }
      set2{
         string effectedBy "Power Plant ID_20"
         double probability 1
         double delay 75.0
         double restore 0.0
         string complexEffect "Power Plant ID_21" "Power Plant ID_25"
      }
   }
}

ebo5{
string name "Air Defense Commander2 Mazar-e-Sharif"
float timeDown 4150
string windicator "none"
string findicator "none"
   effectedSets{
      set0{
         string effectedBy "Airport Mazar-e-Sharif"
         double probability 1
         double delay 0.0
         double restore 10.0
      }
   }
}

ebo6{
string name "Airport Herat"
float timeDown 1280
string windicator "none"
string findicator "none"
   effectedSets{
      set0{
         string effectedBy "Sam Production Sites"
         double probability 1
         double delay 0.0
         double restore 10.0
      }
      set1{
         string effectedBy "Air Defense Commander2 Mazar-e-Sharif"
         double probability 1
         double delay 140.0
         double restore 0.0
      }
   }
}

ebo7{
string name "Work Force ID_20"
float timeDown 4150
string windicator "none"
```

```
string findicator "none"
   effectedSets{
      set0{
         string effectedBy "Power Plant ID_25"
         double probability 1
         double delay 20.0
         double restore 200.0
      }
   }
}
```

This COG.par file shows a Work Force dependant on a Power Plant and a Power Plant dependant on a Work Force. Mazar-e-Sharif Airport is dependant in a complex way on the three Power Plants and the Air Defense Commander2 depends on the Airport.

Sample Assets.par File using WeaponLoads

```
AirAsset{
AirAsset1{
    int Assets 10
    string AssetName "FA-18"
    int ObjType 201
    string Commander "USS Enterprise Battle Group"
    string Reference "USS Enterprise"
    float LatDegrees 24
    float LngDegrees 64
    float AltKm 0
    string ObjAlliance "Friendly"
    float Status 100
    float RadarRange 30
    float JammingRange 20
    float EngagementRange 10
    float DamageThreshold 50
    float MaxSpeed 0.309355
    float MaxDistance 0
}
AirAsset2{
    int Assets 10
    string AssetName "FA-18"
    int ObjType 201
    string Commander "USS Enterprise Battle Group"
    string Reference "USS Enterprise"
    float LatDegrees 24
    float LngDegrees 64
    float AltKm 0
    string ObjAlliance "Friendly"
    float Status 100
    float RadarRange 30
    float JammingRange 20
    float EngagementRange 10
    float DamageThreshold 50
    float MaxSpeed 0.309355
    float MaxDistance 0
    int WeaponLoadID 2
 }
AirAsset3{
    int Assets 5
    string AssetName "C-17"
    int ObjType 240
    string Commander "USAF Northern Commander"
```

```
    string Reference "Qarshi FLD"
    float LatDegrees 38.78
    float LngDegrees 65.75
    float AltKm 5
    string ObjAlliance "Friendly"
    float Status 100
    float RadarRange 30
    float JammingRange 20
    float EngagementRange 10
    float DamageThreshold 70
    float MaxSpeed 0.259355
    float MaxDistance 0
    int WeaponLoadID 3
 }
}

NavalAsset{
NavalAsset1{
    int Assets 1
    string AssetName "Battle Group"
    int ObjType 401
    string Commander "USS Enterprise Battle Group"
    string Reference "USS Enterprise"
    float LatDegrees 24
    float LngDegrees 64
    float AltKm 0
    string ObjAlliance "Friendly"
    float Status 1000
    float RadarRange 50
    float JammingRange 35
    float EngagementRange 10
    float DamageThreshold 100
    float MaxSpeed 0.309355
    float MaxDistance 0
}
}

SAMAsset{
SAMAsset1{
    int Assets 1
    string AssetName "Gadfly"
    int ObjType 501
    string Commander "Air Defense Commander2"
    string Reference "Mazar-e-Sharif"
    float LatDegrees 36.7
    float LngDegrees 67.1
    float AltKm 0
    string ObjAlliance "Hostile"
    float Status 100
    float RadarRange 85
    float JammingRange 10
    float EngagementRange 22
    float DamageThreshold 50
    float MaxSpeed 0.015955
    float MaxDistance 0
}
}

MissileAsset{
}

VehicleAsset{
}

FixedAsset{
FixedAsset1{
    int Assets 1
```

```
    string AssetName "Airport"
    int ObjType 101
    string Reference "Diego Garcia AFB"
    float LatDegrees -7.33
    float LngDegrees 72.42
    float AltKm 0
    string ObjAlliance "Neutral"
    float Status 100
    float RadarRange 50
    float JammingRange 15
    float DamageThreshold 15
 }
FixedAsset2{
    int Assets 1
    string AssetName "Airport"
    int ObjType 101
    string Reference "Herat"
    float LatDegrees 34.33
    float LngDegrees 62.2
    float AltKm 0
    string ObjAlliance "Hostile"
    float Status 100
    float RadarRange 50
    float JammingRange 15
    float DamageThreshold 15
 }
FixedAsset3{
    int Assets 1
    string AssetName "Airport"
    int ObjType 101
    string Reference "Kandahar"
    float LatDegrees 31.62
    float LngDegrees 65.72
    float AltKm 0
    string ObjAlliance "Hostile"
    float Status 100
    float RadarRange 50
    float JammingRange 15
    float DamageThreshold 15
 }
FixedAsset4{
    int Assets 1
    string AssetName "Airport"
    int ObjType 101
    string Commander "Air Defense Commander2"
    string Reference "Mazar-e-Sharif"
    float LatDegrees 36.7
    float LngDegrees 67.1
    float AltKm 0
    string ObjAlliance "Hostile"
    float Status 100
    float RadarRange 50
    float JammingRange 15
    float DamageThreshold 15
 }
FixedAsset5{
    int Assets 1
    string AssetName "Airport"
    int ObjType 101
    string Commander "USAF Northern Commander"
    string Reference "Qarshi FLD"
    float LatDegrees 38.78
    float LngDegrees 65.75
    float AltKm 5
    string ObjAlliance "Friendly"
    float Status 100
    float RadarRange 50
```

```
    float JammingRange 15
    float DamageThreshold 15
 }
FixedAsset6{
    int Assets 1
    string AssetName "Power Plant"
    int ObjType 103
    string Reference "ID_25"
    float LatDegrees 35.40
    float LngDegrees 66.10
    float AltKm 0
    string ObjAlliance "Hostile"
    float Status 100
    float RadarRange 0
    float JammingRange 0
    float DamageThreshold 15
 }
FixedAsset7{
    int Assets 1
    string AssetName "Power Plant"
    int ObjType 103
    string Reference "ID_21"
    float LatDegrees 36.40
    float LngDegrees 67.10
    float AltKm 0
    string ObjAlliance "Hostile"
    float Status 100
    float RadarRange 0
    float JammingRange 0
    float DamageThreshold 15
 }
FixedAsset8{
    int Assets 1
    string AssetName "Power Plant"
    int ObjType 103
    string Reference "ID_20"
    float LatDegrees 37.40
    float LngDegrees 68.10
    float AltKm 0
    string ObjAlliance "Hostile"
    float Status 100
    float RadarRange 0
    float JammingRange 0
    float DamageThreshold 15
 }
FixedAsset9{
    int Assets 1
    string AssetName "Bunker"
    int ObjType 104
    string Reference "ID_19"
    float LatDegrees 31.62
    float LngDegrees 65.7
    float AltKm 0
    string ObjAlliance "Hostile"
    float Status 100
    float RadarRange 0
    float JammingRange 0
    float DamageThreshold 15
 }
}
```

Shows how the SpSets are arranged with respect to each other.

It should be noted that this file displays in the AirAsset section how WeaponLoadID can be omitted, and the simulation work with the default WeaponLoad configuration. It also shows that MissileAsset and VehicleAsset SpSets are empty, but present.

Sample Commanders.par File

```
Objects{
  reference Commander0  // Air Defense Commander2
  reference Commander1  // USAF Northern Commander
  reference Commander2  // USS Enterprise Battle Group
}

Commander0{
   string Commander "Air Defense Commander2"
   string Reference "Mazar-e-Sharif"
Mission1{
string AssetName "Gadfly"
string MissionTime "2:20"
   commands{
      Command001{
        string CommandName "move"
        float Speed 20
        string route "route44"
      }
      Command002{
        string CommandName "operate"
      }
      Command003{
        string CommandName "report"
        string CommanderName "Air Defense Commander2"
      }
   }
 }
}

Commander1{
   string Commander "USAF Northern Commander"
   string Reference "Qarshi FLD"
Mission1{
string AssetName "C-17"
int NewMission 1
int WeaponLoad 3
string MissionTime "{3,  0:20,  1}"
   commands{
      Command001{
        string CommandName "move"
        float Speed 370
        string route "route45"
      }
      Command002{
        string CommandName "engage"
        string Target "Work Force"
        string Reference "ID_21"
        string WeaponDesignation "PW-001"
      }
      Command003{
        string CommandName "move"
        float Speed 350
        string route "route46"
      }
      Command004{
        string CommandName "land"
        string Target "Airport"
```

```
        string Reference "Qarshi FLD"
      }
      Command005{
        string CommandName "report"
        string CommanderName "USAF Northern Commander"
      }
   }
 }
}

Commander2{
   string Commander "USS Enterprise Battle Group"
   string Reference "USS Enterprise"
Mission1{
string AssetName "FA-18"
string MissionTime "{4,  0:10,  2}"
   commands{
      Command001{
        string CommandName "move"
        float Speed 800
        string route "route7"
      }
      Command002{
        string CommandName "engage"
        string Target "Power Plant"
        string Reference "ID_25"
      }
      Command003{
        string CommandName "move"
        float Speed 800
        string route "route32"
      }
      Command004{
        string CommandName "engage"
        string Target "Power Plant"
        string Reference "ID_21"
      }
      Command005{
        string CommandName "move"
        float Speed 700
        string route "route34"
      }
      Command006{
        string CommandName "engage"
        string Target "Power Plant"
        string Reference "ID_20"
      }
      Command007{
        string CommandName "move"
        float Speed 700
        string route "route35"
      }
      Command008{
        string CommandName "move"
        float Speed 700
        string route "route36"
      }
      Command009{
        string CommandName "land"
        string Target "Battle Group"
        string Reference "USS Enterprise"
      }
      Command010{
        string CommandName "report"
        string CommanderName "USS Enterprise Battle Group"

      }
```

```
   }
 }
}

route7{
// Air Enterprise To Mazar-e-Sharif
    Vectors0{
      float       LatDegrees  25.0
      float       LngDegrees  64.1
      float       AltKm  15.0
    }

    Vectors1{
      float       LatDegrees  31.62
      float       LngDegrees  65.72
      float       AltKm  15.0
    }

    Vectors2{
      float       LatDegrees  35.40
      float       LngDegrees 66.10
      float       AltKm  0.5
    }
}
route32{
// Mazar-e-Sharif To Herat
    Vectors0{
      float       LatDegrees  36.0
      float       LngDegrees  66.70
      float       AltKm  1.5
    }

    Vectors1{
      float       LatDegrees  36.40
      float       LngDegrees  67.10
      float       AltKm  0.5
    }
 }
route34{
// Herat ReAttack
    Vectors0{
      float       LatDegrees  36.50
      float       LngDegrees  67.50
      float       AltKm  1.5
    }

    Vectors1{
      float       LatDegrees  37.40
      float       LngDegrees  68.10
      float       AltKm  0.5
    }
 }
route35{
// Herat ReAttack
    Vectors0{
      float       LatDegrees  35.33
      float       LngDegrees  65.20
      float       AltKm  1.5
    }
    Vectors1{
      float       LatDegrees  34.33
      float       LngDegrees  62.20
      float       AltKm  0.5
    }
 }

route36{
```

```
// Herat to Enterprise
    Vectors0{
      float       LatDegrees  33.33
      float       LngDegrees  63.30
      float       AltKm  15
    }

    Vectors1{
      float       LatDegrees  30.33
      float       LngDegrees  63.40
      float       AltKm  15
    }

    Vectors2{
      float       LatDegrees  24
      float       LngDegrees  64.00
      float       AltKm  0.0
    }
 }
route44{
// Mazar-e-Sharif SAM Deployment
    Vectors0{
      float       LatDegrees  36.6999
      float       LngDegrees  67.1001
      float       AltKm  0.0
    }
 }
route45{
// Qarshi to  Mazar-e-Sharif
    Vectors0{
      float       LatDegrees  38.0
      float       LngDegrees  67.0
      float       AltKm  10.0
    }

    Vectors1{
      float       LatDegrees  36.7
      float       LngDegrees  67.1
      float       AltKm  5
    }
 }
route46{
// Mazar-e-Sharif to Qarshi
    Vectors0{
      float       LatDegrees  38.0
      float       LngDegrees  66.9
      float       AltKm  10.0
    }

    Vectors1{
      float       LatDegrees  38.77
      float       LngDegrees  65.72
      float       AltKm  5
    }
 }

RegionOfInterest{
    string TheaterName "SW_AsiaTheater"
}
```

This sample Commanders.par file shows the use of WeaponDesignation in “engage” commands, and default use of Weapons when no designation or WeaponName is employed.

Occasionally, air assets fail during landing. One source of this problem arises because the AirAsset has not discovered the Airport or Battle Group before it executes the land command. One remedy for this problem is to terminate the Vector to the airport/battle group a short distance from the target, at a slight elevation. In this way the MobileAsset discovers the target before it executes the land command.

In this sample Commanders.par file, a SAM asset, Gadfly, is deployed at 2:20 into the simulation, then begins operation and reports to the commander that it is available for a new mission. Three C-17 are launched from Qarshi to drop leaflets on the work force for one of the power plants. They return to Qarshi. Four FA-18s are launched from the Battle Group against the three power plants, each engaging each of the Power Plants.

Note further that the NewMission Field is optional. It appears in the SpSet for the C-17, but not for the FA-18.

Section 6 Running a Simulation

To run FSS, create a directory to hold the simulation configuration (par) files. Move or create the simulation configuration files in that directory. Ensure that fss is in your path. When running on a single node, type:

prompt-> fss

Command line help options (a man-type page) is available with:
prompt> fss –H | more

or

prompt-> fss -? | more

Options include:

-m message_level — message_level is an integer ranging from 1 to 6, the default is 2. This determines how much data is reported to stdout. 1 is a minimum amount of information. 5 includes output detailing whether an asset engaged by an air asset is observable. Experience is the best way to decide what number is best for a particular use.

- 1 Includes Asset Reporting and Process WeaponHits
- 2 Discover/Undiscover Messages; Sign On/Off; EBO Status Changes
- 3 Data map details (Assets/References/Weapon and WeaponLoads
- 4 SAM Sensing reports; Receipt of New Commands; ExecutingMotionCommand

5	AirAsset Observability of target Reports
6	Additions to the SensedAsset List

-s seed_value	seed_value is a number to use as the seed for the random number generator for this simulation.
-d database_name	database_name is the name of the SQL database to access.
-h hostname	hostname is the name of the host that the database resides on.
-g	generate par files.
-C	Format output for post simulation analysis (CASA)

For execution on multiple nodes fss uses the Speedes option:

-lnodes num	number of local process to execute
-nnodes num	number of total process in this simulation.

When running multiple processes, it is necessary to initiate a SpeedesServer process to enable synchronization and communication among the process and also to any external modules that need to connect to the simulation, such as a map based viewer. The SpeedesServer should be running before you start the external module or the instances of fss.

Section 7 Driving FSS with an External Module

For an intelligent adversary to drive the assets in an fss simulation, an external module is required. Several extensions to existing FSS classes have been created and debugged to facilitate this process. A prototype for debugging the external module, which reads a commanders-like.par file for the adversary missions, was investigated as aiix_proto. This prototype made clear the importance of consistency among the various configuration files. The Assets.par file, the COG.par file and the Commanders.par file require a consistency of names, types and references because of the hash tables used to map between names, references, commanders, and their identifying numbers. With the addition of a separate file for adversary missions, consistency and in fact ordering of commanders in the Commanders.par and Commanders-like.par file is critical. Details about this can be found in “Interfacing to FSS from a Speedes External Module.”